МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Озерский технологический институт — филиал НИЯУ МИФИ

Вл. Пономарев

Конспективное изложение теории языков программирования и методов трансляции

Книга 2. Лексический анализ

Учебно-методическое пособие

Озерск, 2019

УДК 681.3.06
П56

Вл. Пономарев. Конспективное изложение теории языков программирования и методов трансляции. Учебно-методическое пособие. В 4-х книгах. Книга 2. Лексический анализ. Озерск: ОТИ НИЯУ МИФИ, 2019. — 48 с., ил.

Книга 2 содержит введение в лексический анализ, теорию регулярных языков и конечных автоматов. Рассматриваются регулярные выражения и грамматики, детерминизация и минимизация конечных автоматов, переход от одних форм представления регулярных языков к другим, свойства регулярных языков.

В качестве вспомогательного материала пособие предназначено для студентов, обучающихся по направлению подготовки 09.03.01 «Информационная и вычислительная техника», и по специальности 09.05.01 «Применение и эксплуатация автоматизированных систем специального назначения».

Рецензенты:
1. Р. Р. Акопян, к.ф.-м.н., зав. кафедрой ПМ ОТИ НИЯУ МИФИ.
2. В. Е. Синяков, начальник УИТ ВГУП «ПО «Маяк».

УТВЕРЖДЕНО
Редакционно-издательским
Советом ОТИ НИЯУ МИФИ

Содержание

Введение

В этой части рассматривается теория регулярных языков и конечных автоматов. Регулярные языки являются наиболее простым типом языков в иерархии Хомского. Цепочки регулярного языка образованы конкатенацией, объединением и итерацией знаков. Конечные автоматы распознают цепочки регулярных языков, анализируя их знак за знаком, вследствие чего скорость распознавания эквивалентна длине цепочки.

Регулярные языки могут быть заданы одним из трех способов: регулярными грамматиками, регулярными выражениями и конечными автоматами. Для практического применения лучше всего подходят регулярные выражения. Они просты, точны и понятны, поэтому используются не только в трансляции языков программирования, но имеют многих других практических приложений.

Конечный автомат также задает регулярный язык, но его назначение заключается в обнаружении цепочек языка, это тот механизм, который на практике реализует регулярное выражение. Конечный автомат легко построить по заданному регулярному выражению. При этом получается недетерминированный конечный автомат с пустыми переходами, реализация которого программно затруднена. Теория конечных автоматов дает математически точный ответ на вопрос, как преобразовать такой автомат в минимально возможный детерминированный распознаватель.

Теория регулярных языков изучает также их свойства. Это позволяет получить ответы на такие важные вопросы, как доказательство эквивалентности языков и автоматов, выявление границ области допустимых операций, определение принадлежности языка к регулярному типу.

Регулярные грамматики редко используются на практике, они скорее нужны для теоретических изысканий. С их помощью, например, можно объединять регулярные языки с контекстно-свободными.

Контекстно-свободные языки описываются подробнее в следующей книге пособия, содержащей теорию синтаксического разбора.

Для самоконтроля усвоения излагаемого материала в конце главы есть вопросы и упражнения, которыми не следует пренебрегать.

Все приведенные в пособии алгоритмы смоделированы и проверены. Теоремы и их доказательства приводятся в отдельной книге.

Текст пособия постоянно совершенствуется, поэтому рекомендуется использовать его электронную версию, которая периодически обновляется на сайте http://revol.ponocom.ru.

2. Лексический анализ

Лексический анализ — это процесс выделения во входной цепочке отдельных простейших смысловых конструкций языка — *лексем*.

Целью лексического анализа является обнаружение лексем входного текста, их выделение, категоризацию, преобразование в удобную для последующего анализа форму (*токен*), и формирование потока токенов вместо потока символов (рисунок 2.1).

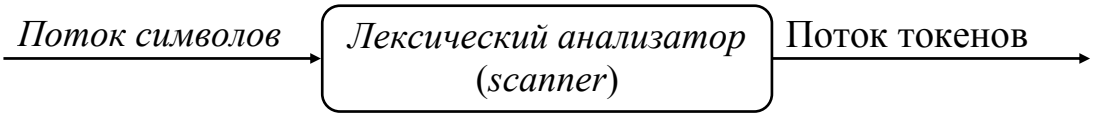


Рисунок 2.1. Лексический анализ

В следующем за лексическим анализе синтаксическом разборе поток токенов является входным текстом, а сами токены являются терминальными символами синтаксических грамматик.

Лексемы описываются *регулярными языками*, имеющими самый простой тип. Регулярные языки, в свою очередь, являются *регулярными множествами*. Для задания регулярных множеств используются *регулярные грамматики*, *регулярные выражения* и *конечные автоматы*.

Регулярные выражения являются простой и удобной формой записи регулярных множеств. Конечные автоматы используются для разработки лексических анализаторов.

Доказано, что все три способа задания регулярных языков (регулярных множеств) являются эквивалентными, и существуют формальные методы перехода от одной формы представления к другой.

2.1. Применения лексического анализа

Лексический анализ является составной частью процесса трансляции языка программирования. Однако применение лексических анализаторов не ограничивается только языками программирования. Следующие примеры показывают практические приложения теории конечных автоматов, регулярных языков и лексического анализа.

1. Для конструирования цифровых схем, составляющих аппаратную платформу компьютеров, и не только компьютеров, используют конечные автоматы, моделирующие вычислительные устройства.

2. В текстовых редакторах поиск и замена слов использует механизм регулярных выражений, называемый также подстановочными знаками.

3. В операционных системах регулярные выражения задают маски файлов (*wildcards*), и используются для поиска и замены текста.

4. Несмотря на то, что лексический анализатор уже является частью языка программирования, в некоторые языки встроены распознаватели регулярных выражений для формирования лексических анализаторов «на лету». Эти распознаватели выполняют разбор цепочек, которые должны контролироваться не во время создания программы, а во время ее работы. Контролируемые цепочки при этом имеют отношение не к собственно языку программирования, а к предметной области, в которой работает транслятор.

5. Конечные автоматы используются для реализации или контроля систем, которые имеют конечное число состояний. Такими системами являются, например, протоколы связи и обмена информацией.

Эти и другие примеры убеждают в необходимости всестороннего изучения теории регулярных языков и конечных автоматов.

2.2. Лексемы языков программирования

Рассмотрим лексемы языков программирования, и классифицируем их с той или иной точки зрения.

2.2.1. Категории лексем

В языках программирования встречаются следующие категории, или виды лексем с точки зрения их целевого назначения: идентификатор, ключевое слово, литерал (константа), символ или знак операции, символ или знак пунктуации (пунктуатор), комментарий.

Идентификатор — это последовательность знаков, начинающаяся с буквы, за которой может следовать любое количество букв и цифр. Часто начальная и последующая часть включает также знак подчеркивания.

Ключевые слова являются идентификаторами, но отличаются тем, что их длина, в отличие от длины идентификатора, точно известна.

Литерал — это непосредственная запись значения (литерами). Различают числовые, символьные и строковые литералы.

Числовые литералы делятся на целочисленные и вещественные. Числовые литералы записываются в одной из систем счисления: двоичной, восьмеричной, шестнадцатеричной или десятичной.

Символьные и строковые литералы заключают в кавычки, двойные и одинарные. Они состоят из литер (букв, цифр, знаков), но могут включать в себя специальные последовательности.

Знаки операций — это обычно один или два знака. Иногда операции обозначают ключевыми словами, например div, mod, and, or, xor и т.п.

Пунктуаторы, — это все возможные скобки, точка, запятая, точка с запятой, двоеточие и т.п. Они состоят из одного, редко из двух знаков.

2.2.2. Обнаружение и выделение лексем

Обнаружение подразумевает определение типа очередной лексемы входного потока, а выделение — это определение ее границ.

В современных языках программирования тип лексемы можно определить по ее первому знаку. Например, идентификатор начинается со знака подчеркивания или буквы, числовой литерал начинается с цифры или точки, строковый литерал начинается с двойной кавычки, символьный литерал — с одинарной кавычки.

С точки зрения границ лексемы делятся на следующие виды.

1. Лексемы, границы которых заданы предопределенными знаками или последовательностями знаков. К ним относятся строковые и символьные литералы и комментарии. Левая граница этих лексем обнаруживается просто, однако правая граница не всегда заканчивает лексему. Например, строковый литерал может содержать в себе тот же знак, что и ограничивающий лексему.

2. Лексемы, которые заканчиваются первым недопустимым знаком. Это идентификаторы и числовые литералы. Проблема идентификатора в том, что во время распознавания недопустимый знак уже обнаружен, однако он может входить в состав уже другой лексемы.

3. Лексемы заданной длины. К ним относятся ключевые слова, операции и пунктуаторы. Правая граница этих лексем определяется просто, а начало лексемы часто определяется по принципу — все, что не идентификатор или литерал, есть лексема заданной длины.

Лексемы разделяют также пробельные символы (*whitespaces*), которые не входят в состав лексем и никак не учитываются. К ним относятся собственно пробел, табуляторы, символ новой строки. Символ новой строки, однако, в некоторых языках является лексемой-пунктуатором.

2.2.3. Структура лексем

С точки зрения распознавания важна структура лексем.

Входной поток — это знаки. Некоторые знаки однотипны в том смысле, что при распознавании их можно отнести к одному типу. Такими типами являются буквы, цифры и недопустимые знаки. Поэтому есть возможность распознавать не только знаки, но и псевдо-знаки, или заменители. Знаки преобразуются в заменители таблицей перекодировки.

Для распознавания одних лексем требуется анализ знаков, а для распознавания других достаточно анализировать псевдо-знаки. Лексемы часто таковы, что приходится сочетать анализ знаков и заменителей. С этой точки зрения лексемы языков программирования могут иметь одну из следующих структур.

1. Лексемы конечной длины, описываемые конечными языками. К ним относятся ключевые слова, знаки операций и пунктуаторов, символьные литералы. Эти лексемы формируются конкатенацией знаков, и хотя они очень просты, алфавит распознавания получается широким. Это усложняет формулу языка и структуру распознавателя.

2. Лексемы, длина которых, в принципе, ничем не ограничивается. К ним относятся идентификаторы и литералы, за исключением символьных. Эти лексемы описываются бесконечными языками, основанными на итерации заменителей, поэтому алфавит распознавания узкий. Формула языка и структура распознавателя получаются простыми.

Лексемы второго типа часто описываются и бесконечными, и конечными языками, например, таковы вещественные константы, имеющие самую сложную структуру.

2.3. Распознавание лексем

Лексемы распознаются конечными автоматами.

Конечные автоматы, или просто автоматы или машины, появились в 40-х годах XX века, и первоначально с их помощью моделировали функционирование человеческого мозга. Впоследствии для них нашлись и другие интересные применения.

Автомат реализует циклический алгоритм, в каждой итерации которого очередной символ входной цепочки проверяется на соответствие с заданным шаблоном (порядком). Автоматы «помнят» свое состояние и изменяют его при наступлении некоторого события. При распознавании лексем события — это знаки входной цепочки, последовательно поступающие на вход.

2.3.1. Состояния распознавания

Ключевым аспектом распознавания является *понятие состояния*.

Прежде рассмотрим модель электрического выключателя, имеющего два состояния, «выключено» и «включено», причем первое состояние является исходным, начальным. Нажатие кнопки выключателя меняет состояние устройства на противоположное. Эту модель удобно изобразить в виде графа, как показано на следующем рисунке.

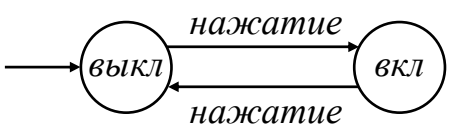


Рисунок 2.2. Граф переходов системы с двумя состояниями

Состояния на рисунке обозначены кружками. Начальное состояние указывается направленной на него стрелкой, которую обычно помечают словом *старт* или *начало*. Возможные переходы между состояниями задаются направленными ребрами (стрелками). Метка «*нажатие*» возле ребра обозначает событие (*event*), ведущее к изменению состояния.

Сначала модель находится в состоянии, помеченном «*выкл*». Модель «*помнит*» свое текущее состояние *фактом нахождения в нем*. Поэтому событие «*нажатие*» переводит модель в состояние «*вкл*», как указывает верхняя стрелка, и модель «*запоминает*» свое новое текущее состояние.

Рассмотрим теперь, как можно распознать лексемы типа «ключевое слово», например, слова "*for*" и "*if*". Распознавать будет некий алгоритм, процесс распознавания изобразим в виде графа переходов, как показано на следующем рисунке.

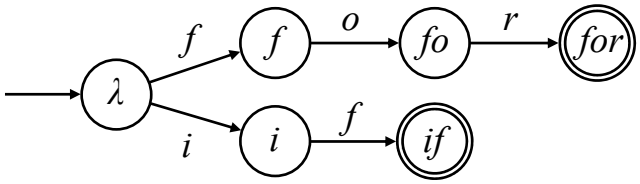


Рисунок 2.3. Распознавание слов *for* и *if*

Первоначально алгоритм находится в исходном состоянии, которое характеризуется готовностью к анализу. Кроме того, алгоритм каким-то образом получает событие в виде текущего знака входной цепочки.

Пусть событие равно "*f*". Тогда можно предположить, что ожидается цепочка "*for*", и перейти к состоянию, которое «*помнит*», что распознан знак "*f*". Нужно также получить следующий знак.

Если следующий знак "*o*", алгоритм переходит в состояние, которое «*помнит*», что распознаны уже два знака "*fo*". Наконец, получив знак "*r*", алгоритм переходит в состояние, которое «*помнит*», что распознаны знаки "*for*", то есть все слово. Это состояние отмечено двойной рамкой, соответствует завершению распознавания, называется заключительным, финальным или допускающим. В допускающем состоянии алгоритм как-то сигнализирует о том, что цепочка допускается как искомая.

Заметим, что у данного алгоритма допускающих состояний два.

Состояния принято как-то обозначать. Один из вариантов нумерует состояния числами, а другой буквами. Мы пока обозначили состояния распознанной цепочкой. Так, в начальном состоянии еще не распознано ни одного знака, поэтому состояние обозначено как λ, пустая цепочка. В последнем состоянии слово распознано, и оно обозначено как "*for*".

Слово "*if*" распознается аналогично. Заметим, что в этом случае текущий знак "*f*" переводит алгоритм в состояние "*if*", а не в состояние "*f*".

2.3.2. Функция переходов

До сих пор мы говорили о распознавании лексемы или цепочки при помощи некоторого алгоритма. При этом мы опустили вопрос, откуда алгоритм «знает», какое состояние является следующим, если он помнит только текущее? Если текущий знак "*f*", почему алгоритм переходит в одном случае в состояние "*f*", а в другом случае в состояние "*if*"?

Алгоритм станет автоматом, когда мы зададим *порядок переходов*, соответствующий стрелкам графа. Определим для этой цели функцию δ(q, *a*), где q — это текущее состояние, а *a* — текущий знак (событие). Значением функции δ(q, *a*) является то состояние, в которое переходит алгоритм, находясь в состоянии q при наступлении события *a*.

Для графа на рисунке 2.3 функция дельта может быть определена следующим образом.

Функции для распознавания цепочки "*for*":

δ("*λ*", *f*) = "*f*"

δ("*f* ", *o*) = "*fo*"

δ("*o*", *r*) = "*for*"

Функции для распознавания цепочки "*if* ":

δ("*λ*", *i*) = "*i*"

δ("*i*", *f*) = "*if*"

Алгоритм работы автомата можно теперь схематически изобразить примерно так, как показано на следующем рисунке.

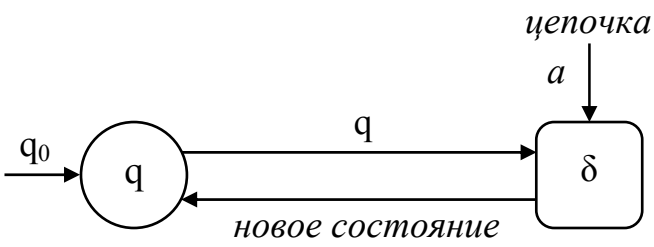


Рисунок 2.4. Схема распознающего автомата

Автомат начинает работу в состоянии q_0, которое запоминается как текущее в блоке q. Как только получен текущий знак *a*, функция переходов δ вырабатывает новое состояние, которое становится текущим.

2.3.3. Крайнее состояние

Мы также опустили вопрос, в какое состояние переходит автомат, если он, например, находится в состоянии "*fo*", но следующее событие не знак "*r*", то есть во входном тексте слово не "*for*"? Теория автоматов не дает на этот вопрос подходящего для программиста ответа.

Автоматы распознают только правильные цепочки, а неправильные должны отвергать, но как это сделать теория умалчивает. Тем не менее, с точки зрения программирования все не так плохо. Алгоритм автомата, например, может быть реализован по следующей схеме:

```
q = q0
повторять {
    получить x
    q = δ(q, x)
    если q допускающее, сигнализировать "допуск", конец
    если q пусто, сигнализировать "ошибка", конец
}
```

Когда автомат переходит в допускающее состояние, функция, реализующая алгоритм, завершается и возвращает истину. Если же функция переходов не определена, функция алгоритма завершается, возвращая в этом случае ложь.

По теории мы можем определить переходы, которые во всех неясных случаях ведут в *крайнее* (*мертвое*, *фиктивное*, *wild*) состояние, в котором как бы собираются ошибки распознавания. Это состояние не может быть допускающим, хотя по смыслу является финальным, так как оно должно завершать работу алгоритма.

При попадании в *крайнее* состояние согласно теории автомат остается в нем, совершая переходы в него же по всем возможным символам. С точки зрения теории автоматы не являются функциями, которые могут что-то возвращать, и понятие завершения работы для них неприменимо.

Рассмотрим пример с ключевыми словами (рисунок 2.3). Добавим в граф еще одно состояние, в которое направим ход распознавания во всех ошибочных случаях. Нам нужно также определить алфавит Σ, который содержит все возможные знаки. Полностью определенный граф переходов показан на следующем рисунке. Крайнее состояние помечено *Z*.

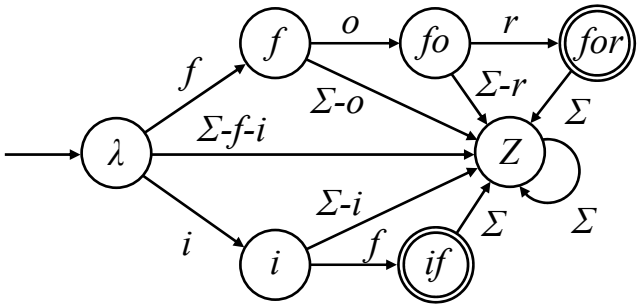


Рисунок 2.5. Полностью определенный граф переходов

Говорят, что в крайнем состоянии автомат тихо «*умирает*», так как в итоге не сможет сделать ни одного движения из *Z*. Если же крайнее состояние не предусмотрено, автомат просто как-то «*умирает*».

2.3.4. Распознаваемые языки

Неоценимый вклад в теорию конечных автоматов и регулярных языков внес американский математик и логик *Стивен Коул Клини*. Именно он является автором регулярных выражений и операции «звездочка», названной в его честь «*звездочкой Клини*». В 1956 г. Клини показал, что язык является регулярным тогда и только тогда, когда он допускается конечным автоматом, а языки, распознаваемые конечными автоматами, совпадают с языками, заданными регулярными выражениями.

Допускаемый автоматом язык — это множество всех допускаемых цепочек. Для графа на рисунке 2.3 это множество состоит только из слов "*for*" и "*if*". Язык, который при этом распознается, образован конкатенацией знаков (в определенном порядке).

Такой язык был бы весьма бедным, потому что конкатенация создает только жестко заданные последовательности знаков, то есть только определенные слова. По счастью, автоматы допускают и более сложные структуры цепочек, в чем можно убедиться на следующем примере.

Рассмотрим граф, показанный на рисунке 2.6.

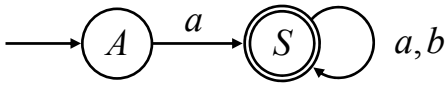


Рисунок 2.6. Граф переходов распознавателя идентификатора

Здесь из состояния *S* автомат переходит в это же состояние. И хотя стрелка *S*→*S* на графе одна, ей соответствует две функции переходов, и можно было бы нарисовать на графе две стрелки, помеченные отдельно.

Интуитивно понятно, что этот автомат допускает цепочки, которые могут содержать бесконечное число знаков, причем первый знак любой допустимой цепочки — это знак *a*, и за ним следует последовательность знаков *a* и *b* в произвольном порядке.

Если принять, что *a* — это буква, а *b* — это цифра, то этот автомат распознает идентификаторы вида *буква* { *буква* | *цифра* }. Распознаваемый им язык определяется при помощи еще двух операций.

Первая операция *объединяет* знаки *a* и *b*, и это означает, что из двух знаков можно, проходя по стрелке, выбрать любой один.

Вторая операция *итерация*, она зацикливает путь распознавания на состоянии *S*, и означает произвольное число конкатенаций объединения знаков *a* и *b* с уже распознанной цепочкой.

Таким образом, языки, распознаваемые конечными автоматами, образованы при помощи *конкатенации*, *объединения* и *итерации*.

Далее мы рассмотрим автоматы и языки более формально.

2.4. Регулярные множества и выражения

2.4.1. Операции над регулярными языками

В контексте регулярных языков только три операции над языками интересны нам: объединение, конкатенация и итерация. Эти операции называют также *регулярными операциями* или *операторами*.

Пусть задан алфавит Σ и языки $L(\Sigma)$ и $M(\Sigma)$.

1. *Объединение* языков L и M определяется как множество цепочек, выводимых из обоих языков:

$L \cup M = \{ \alpha \mid \alpha \in L$ или $\alpha \in M \}$.

Объединение ассоциативно и коммутативно.

2. *Конкатенация* языков L и M определяется как множество цепочек, в которых цепочки языка M приписаны после цепочек языка L:

$LM = \{ \alpha\beta \mid \alpha \in L, \beta \in M \}$.

Конкатенация ассоциативна, но не коммутативна.

3. *Замыкание* (*closure*) или итерация — это повторение цепочек языка множество раз. *Транзитивное* замыкание L^* (иначе замыкание *Клини*, звездочка *Клини*, операция «*звездочка*») определяется как конкатенация языка с самим собой нуль или более раз:

$L^* = \{ \alpha^n \mid \alpha \in L, n \geq 0 \}$.

Заметим, что L^0 обозначает нуль повторений, то есть $L^0 = \{\lambda\}$.

L^1 обозначает одно повторение каждой цепочки, поэтому $L^1 \equiv L$.

L^2 обозначает повторение каждой цепочки языка дважды.

L^* обозначает любое число повторений, хоть 0, хоть 1, хоть 2 и т.д.

Позитивное замыкание L^+ определяется как конкатенация языка с самим собой один или более раз:

$L^+ = \{ \alpha^n \mid \alpha \in L, n > 0 \}$.

Пример 2.*а*. Операции над регулярными языками

Пусть заданы языки $A = \{A, B, \ldots, Z, a, b, \ldots, z\}$ и $B = \{0, 1, \ldots, 9\}$.

При помощи регулярных операций из языков A и B можно создавать новые языки, которые *также будут регулярными*:

A^* — множество цепочек из нуля или более букв;

AB — множество цепочек из двух символов — буква и цифра;

$A \cup B$ — множество всех латинских букв и цифр;

$(A \cup B)^*$ — множество цепочек из нуля и более букв и цифр;

$A(A \cup B)^*$ — множество цепочек, начинающихся с буквы, за которой следует нуль и более букв и цифр (идентификатор);

B^+ — множество цепочек из одной или нескольких цифр.

BB^* эквивалентно B^+ (целое число без знака).

2.4.2. Регулярные множества

Регулярные множества (и регулярные языки) представляют собой цепочки символов над заданным алфавитом, полученные при помощи регулярных операций. Регулярное множество над алфавитом Σ задается рекурсивно следующим образом:

Базис:

1) $\varnothing$ — регулярное множество;

2) $\{\lambda\}$ — регулярное множество;

3) $\{a\}$ — регулярное множество $\forall a \in \Sigma$;

Индукция: если A и B — это регулярные множества, то:

4) $A \cup B$ — регулярное множество;

5) AB — регулярное множество;

6) A^* — регулярное множество.

7) Ничто другое не является регулярным множеством.

2.4.3. Регулярные выражения

Регулярные выражения задают (описывают) регулярные множества над алфавитом Σ рекурсивно следующим образом.

Базис:

1) 0 — регулярное выражение, обозначающее множество $\varnothing$;

2) 1 — регулярное выражение, обозначающее множество $\{\lambda\}$;

3) a — регулярное выражение, обозначающее множество $\{a\}$ $\forall a \in \Sigma$;

Индукция: если a и b — это регулярные выражения, обозначающие соответственно регулярные множества A и B, то:

4) $a+b$ — регулярное выражение, обозначающее множество $A \cup B$;

5) ab — регулярное выражение, обозначающее множество AB;

6) a^* — регулярное выражение, обозначающее множество A^*;

7) Если E — регулярное выражение, то регулярное выражение (E) задает то же регулярное множество, что и E.

8) Ничто другое не является регулярным выражением.

Соотношение между регулярными множествами и выражениями следующее: конкретное регулярное выражение обозначает только одно регулярное множество, а конкретное регулярное множество может быть задано различными регулярными выражениями.

Приоритет операций для регулярных выражений таков:

1) итерация (высший), 2) конкатенация, 3) объединение (низший).

Операции левоассоциативны (выполняются слева направо).

Изменить порядок операций можно при помощи скобок. Согласно этим правилам, записи $(a)+((b)^*(c))$ и $a+b^*c$ являются эквивалентными.

Пример 2.*б*. Регулярные выражения

Если $\Sigma = \{a, b\}$, то следующие выражения описывают языки:

$(a+b) = \{ a, b \}$;

$(a+b)(a+b) = \{ aa, ab, ba, bb \}$;

$a^* = \{ \lambda, a, aa, aaa, \ldots \}$;

$(a+b)^* = \lambda$ + любая последовательность из a и b;

$(a^*+b^*) = \{ \lambda, a, aa, aaa, \ldots, b, bb, bbb, \ldots \}$;

$(a^*+b^*)^* = \lambda$ + любая последовательность из a и b;

$a+a^*b = \{ a, b, ab, aab, aaab, aaaab, \ldots \}$.

$a^*+a^*b = \{ \lambda, a, b, ab, aa, aab, aaa, aaab, aaaa, aaaab, \ldots \}$.

Так как написание регулярного выражения может совпадать со знаками, например, 0 или 1, может возникнуть путаница. Условимся, что выражения записываются как есть, а знаки при неоднозначности пишем либо в кавычках, либо жирным шрифтом.

2.4.4. Свойства регулярных выражений

Пусть a, b, c — регулярные выражения.

Тогда справедливы следующие тождества:

1. $a+b = b+a$
2. $(a+b)+c = a+(b+c) = a+b+c$
3. $a+a = a$
4. $0+a = a+0 = a$
5. $1a = a1 = a$
6. $0a = a0 = 0$
7. $(ab)c = a(bc) = abc$
8. $a(b+c) = ab+ac$
9. $(b+c)a = ba+ca$
10. $1^* = 1$
11. $0^* = 1$
12. $aa^* = a^*a$
13. $a+a^* = a^*$
14. $(a^*)^* = a^{**} = a^*a^* = a^*$
15. $1+a^* = a^*+1 = a^*$
16. $1+aa^* = 1+a^*a = a^*$
17. $(ab)^*a = a(ba)^*$
18. $(a^*b^*)^* = (a+b)^*$
19. $(a^*b)^*a^* = (a+b)^*$
20. $(a^*+b^*)^* = (a+b)^*$
21. $(a^*b)^* = (a+b)^*b+1$
22. $(ab^*)^* = a(a+b)^*+1$

2.5. Конечные автоматы

Конечный автомат (КА, *finite automation*, *FA*), — это распознаватель цепочек регулярных языков. Схематически он соответствует одностороннему распознавателю, в котором нет рабочей памяти (рисунок 2.7).

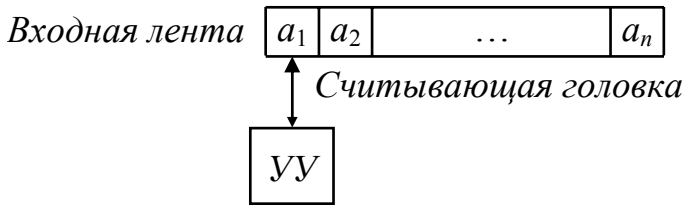


Рисунок 2.7. Схема конечного автомата

2.5.1. Формальное описание

Формально конечный автомат — это пятерка $M = (Q, \Sigma, \delta, q_0, F)$, где

Q — конечное множество состояний,

Σ — конечное множество входных символов (входной алфавит),

δ — функция переходов, отображающая $\Sigma \times Q$ в Q,

q_0 — начальное состояние, $q_0 \in Q$,

F — непустое множество допускающих состояний, $F \subseteq Q$.

Автомат может иметь множество начальных состояний $I \subseteq Q$.

В каждый момент автомат находится в некотором состоянии $q \in Q$, называемом текущим состоянием, и обозревает некоторый символ $a \in \Sigma$ входной цепочки $w \in \Sigma^*$, называемый текущим символом.

Запись $r \in \delta(q, a)$, $r, q \in Q$, $a \in \Sigma$ называют переходом. Он означает, что при текущем состоянии q и текущем символе *a* автомат может перейти в состояние r, сдвинув считывающую головку вправо. Множество всех возможных переходов — это функция переходов.

Пара (q, v), где q — текущее состояние, а *v* — нераспознанная часть цепочки *w*, называется конфигурацией. Начальной конфигурацией является (q_0, w), а допускающей — одна из (f, λ), $f \in F$.

Шаг или такт работы автомата, — это бинарное отношение $\vdash$, определенное на множестве конфигураций следующим образом:

$(q, av) \vdash (r, v)$, если $r \in \delta(q, a)$.

Путь — это последовательность *k* шагов $(q, a_1a_2\ldots a_kv) \vdash^* (r, v)$, $k \geq 0$.

Путь называется успешным, если $q = q_0$, $r \in F$, $v = \lambda$.

Конечный автомат *допускает* цепочку *w* на входе, если для нее есть некоторый успешный путь. Иначе цепочка отвергается. Множество всех допускаемых цепочек есть распознаваемый автоматом язык $L(M)$:

$L(M) = \{ w \in \Sigma^* \mid \exists (q_0, w) \vdash^* (f, \lambda),\ f \in F \}$.

2.5.2. Полный конечный автомат

Конечный автомат называется *полным* (*полностью определенным*), если функция его переходов определена для любого символа и любого состояния: $\forall a \in \Sigma$, $\forall q \in Q$: $\delta(q, a) \neq \varnothing$ (является полной). Иначе конечный автомат называется *неполным* или *не полностью определенным*.

Полный автомат всегда можно получить из неполного, если ввести крайнее состояние, и направить в него все неопределенные переходы. В смысле распознаваемого языка полный автомат остается эквивалентным неполному. Для распознавания и допускания цепочки крайние переходы не важны, автомат в любом случае «*умирает*», если распознавание заходит в тупик, независимо от того, в каком состоянии.

Для примера, автомат, соответствующий графу на рисунке 2.3, неполный, в то время как автомат, соответствующий графу на рисунке 2.5, является полным, при этом они распознают один и тот же язык.

2.5.3. Граф и таблица переходов

Граф переходов и таблица переходов конечного автомата являются наглядными формами его описания. Это соответственно графическое и табличное представление функции переходов δ.

Граф переходов — это помеченный направленный граф, вершины которого обозначены символами состояний $q \in Q$.

Если существует переход $r \in \delta(q, a)$, $r,q \in Q$, $a \in \Sigma$, то на графе между вершинами r и q располагается дуга (ребро), направленная из состояния q в состояние r, и помеченная символом a. Если существует несколько переходов $r \in \delta(q, a_1)$, $r \in \delta(q, a_2)$, ..., $r \in \delta(q, a_n)$, $n > 1$, то дуга помечается всеми символами $a_1, a_2, ..., a_n$, либо рисуется n дуг, помеченными символами a_i, $1 \leq i \leq n$. Первый способ предпочтительнее и используется чаще всего.

Начальные состояния на графе переходов указывают стрелкой извне, помеченной словом «*старт*» или «*начало*», а допускающие состояния отмечают двойной рамкой.

В таблице переходов строки помечены символами состояний $q \in Q$, а столбцы помечены символами $a \in \Sigma$, и, возможно, λ.

Если существует переход $r \in \delta(q, a)$, то состояние r записывается в строку таблицы, помеченную q, и в столбец, помеченный a. Если есть несколько переходов $r_1 \in \delta(q, a)$, $r_2 \in \delta(q, a)$, ..., $r_n \in \delta(q, a)$, $n > 1$, то в строку q и в столбец a записывается множество $\{r_1, r_2, ..., r_n\}$.

Начальные состояния в таблице переходов указывают стрелкой, а допускающие состояния отмечают звездочкой.

В качестве примера рассмотрим конечный автомат M1, распознающий комментарий вида /* ... */ в языке Си. Формальное описание автомата M1 может быть следующим:

M1 = ({*H*, *A*, *B*, *C*, *S*}, {/, *, *a*}, δ, *H*, {*S*}), δ = {
δ(*H*, /) = *A*, δ(*A*, *) = *B*,
δ(*B*, /) = *B*, δ(*B*, *a*) = *B*, δ(*B*, *) = *C*,
δ(*C*, *) = *C*, δ(*C*, *a*) = *B*, δ(*C*, /) = *S*
}.

На следующем рисунке изображен граф переходов автомата M1.

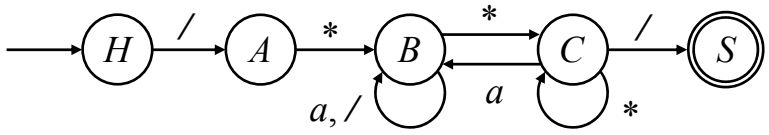


Рисунок 2.8. Граф переходов автомата M1

В таблице 2.1 отображена функция переходов автомата M1.

Таблица 2.1. Таблица переходов автомата M1

M1	/	*	*a*
→*H*	*A*	∅	∅
A	∅	*B*	∅
B	*B*	*C*	*B*
C	*S*	*C*	*B*
**S*	∅	∅	∅

Данный автомат является неполным.

Заметим, что символ *a* — это любой знак входного алфавита, за исключением / и *. Интересны здесь только состояния *B* и *C*, в них автомат принимает внутреннюю часть комментария.

2.5.4. Детерминированный конечный автомат

Некоторый конечный автомат $D = (Q, \Sigma, \delta, q_0, F)$ с одним начальным состоянием $q_0 \in Q$, является *детерминированным* (ДКА, *deterministic FA*, *DFA*), если $\forall q \in Q$, $\forall a \in \Sigma$: либо $\delta(q, a) = r$, либо $\delta(q, a) = \varnothing$, то есть $\delta(q, a)$ возвращает *не более одного* состояния для любых q и *a*.

Будем называть некоторый ДКА *строго детерминированным*, или СДКА, если $\forall q \in Q$, $\forall a \in \Sigma$: $\delta(q, a) = r$. Иначе говоря, его функция $\delta(q, a)$ возвращает *строго одно* состояние для любых q и *a*.

СДКА — это полностью определенный (полный) ДКА.

ДКА является, например, М1. Однако чисто технически автомат М1 не детерминирован в строгом смысле, так как его функция переходов неполная. Точно так же неполный автомат, распознающий цепочки *for* и *if*, и соответствующий графу на рисунке 2.3, является ДКА, в то время как полный автомат, распознающий цепочки *for* и *if*, и соответствующий графу на рисунке 2.5, является СДКА.

Обычно не строго детерминированный конечный автомат полностью устраивает нас, и мы будем обозначать его как ДКА в этом случае и в случае, когда неважно, является ли ДКА строгим или нет.

Таблица переходов СДКА отличается тем, что в каждой ее ячейке записано обозначение *одного* состояния. Таблица переходов ДКА может содержать ячейки, в которых ничего не записано, или записано «пусто».

Детерминированный конечный автомат отличается также тем, что в каждый момент существования он находится ровно в одном состоянии.

2.5.5. Недетерминированный конечный автомат

Недетерминированный конечный автомат (НКА, *nondeterministic FA*, *NFA*) отличается тем, что его функция переходов δ возвращает подмножество $S \subseteq Q$, которое содержит нуль, одно, или несколько состояний. Поэтому НКА может находиться сразу в нескольких состояниях.

На следующем рисунке приведен граф переходов НКА, распознающего цепочки вида $(a+b)^*ab$ [11].

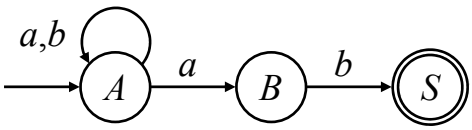


Рисунок 2.9. Граф переходов НКА М2

Функция переходов этого НКА имеет следующий вид::

$\delta(A, a) = \{A, B\}$,

$\delta(A, b) = \{A\}$,

$\delta(B, b) = \{S\}$.

В таблице переходов НКА ячейки также содержат множества (таблица 2.2). Вместо пустых скобок {} в ней можно использовать знак ∅.

Таблица 2.2. Таблица переходов НКА М2

М2	*a*	*b*
→*A*	{*A*, *B*}	{*B*}
B	{}	{*S*}
**S*	{}	{}

Недетерминированность проявляется в состоянии A, которое при событии "a" переводит автомат в состояния A и B. Путь распознавания цепочки "aab" этим автоматом показан на следующем рисунке.

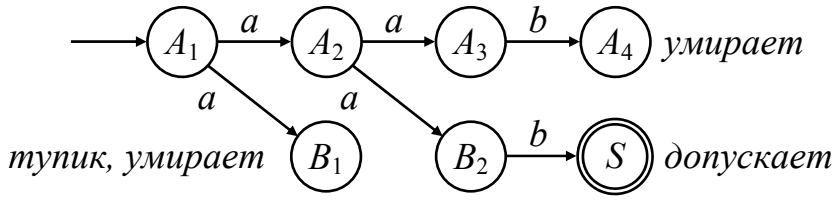


Рисунок 2.10. Распознавание цепочки "aab" НКА M2

Из состояния A_1 по событию "a" НКА переходит в состояния A_2 и B_1. Состояние B_1 тупиковое, — следующее событие "a" не задано функцией переходов, и B_1 «*умирает*». Из состояния A_2 по событию "a" НКА переходит в состояния A_3 и B_2. Далее идет две ветки распознавания, и ветка B_2 достигает допускающего состояния, а ветка A_3 *умирает*.

НКА допускает цепочку w, если для нее существует успешный путь, даже если некоторые пути при этом могут «*умирать*».

Построить НКА обычно проще, чем соответствующий ДКА. Однако НКА сложно моделировать. Поэтому важен вопрос, как получить ДКА по имеющемуся НКА. Доказано, что любому НКА соответствует ровно один эквивалентный ему ДКА.

В основе метода лежит идея о том, что каждому состоянию ДКА соответствует некоторое множество состояний НКА, и переходы из одного состояния по одному и тому же символу ведут не в разные состояния НКА, а во множество состояний, принимаемое равным состоянию ДКА.

2.5.6. Построение ДКА по НКА

Пусть задан некоторый НКА $N = (Q, \Sigma, \delta, I, F)$.

Построим такой ДКА $D = (Q', \Sigma, \delta', q_0, F')$, для которого $L(D) = L(N)$.

1. Множество состояний ДКА Q' сконструируем как множество всех подмножеств состояний Q, то есть булеан 2^Q множества Q. Этот процесс называется конструкцией подмножеств (*powerset construction*). Для НКА с n состояниями ДКА будет иметь 2^n состояний. Это плохая весть, но есть надежда, что не все состояния ДКА будут востребованы.

2. Функцию переходов ДКА построим следующим образом:

$\forall S \subseteq Q'$: $\delta'(S, a) = \bigcup \delta(p, a)$, $\forall p \in S$, $\forall a \in \Sigma$.

Иначе говоря, переходы состояния S равны объединению переходов всех состояний p, входящих в подмножество S.

3. Начальным состоянием ДКА q_0 становится подмножество I, включающее в себя все начальные состояния НКА.

4. Состояние ДКА *S*, включающее в себя хотя бы одно допускающее состояние НКА, становится допускающим состоянием ДКА.

Конструкция подмножеств НКА M2 отображена в таблице 2.3. Будем обозначать состояния ДКА конкатенацией обозначений состояний НКА в квадратных скобках, чтобы подчеркнуть их единственность.

Таблица 2.3. Конструкция подмножеств ДКА M2

M2	*a*	*b*	
∅	∅	∅	×
→[*A*]	[*AB*]	[*A*]	
[*B*]	∅	[*S*]	×
*[*S*]	∅	∅	×
[*AB*]	[*AB*]	[*AS*]	
*[*AS*]	[*AB*]	[*A*]	
*[*BS*]	∅	[*S*]	×
*[*ABS*]	[*AB*]	[*AS*]	×

Построенный ДКА является полным, то есть СДКА. Достижимые состояния только начальное состояние *A* и состояния *AB* и *AS*. Недостижимые состояния отмечены крестиками справа от таблицы.

Удаляя строки с недостижимыми состояниями, и заменяя обозначения состояний, получим не строгий ДКА M2, граф переходов которого показан на следующем рисунке.

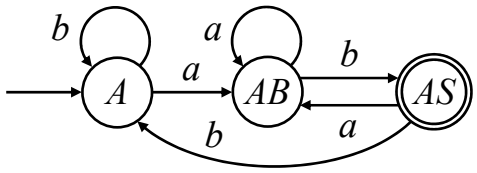


Рисунок 2.11. Граф переходов ДКА M2

Можно построить ДКА без образования недостижимых состояний. Для этого таблица переходов ДКА заполняется строка за строкой по мере образования новых состояний. Следующий алгоритм строит по НКА $N = (Q, \Sigma, \delta, I, F)$ не обязательно строгий ДКА $D = (Q', \Sigma, \delta', q_0, F')$.

Алгоритм 2.1. Построение ДКА по НКА.

Шаг 1. Начальное состояние ДКА принимаем равным объединению начальных состояний НКА: $q_0 = [s_1 \ldots s_j \ldots s_k]$, $\forall s_j \in I$.

Шаг 2. Поочередно рассматриваем новые состояния ДКА. Пусть состояние ДКА обозначено меткой p и равно $[t_1 \ldots t_j \ldots t_k]$. Для каждого символа *a* формируем состояние ДКА, равное объединению переходов $\delta(t_j, a)$, обозначаем его меткой q. Записываем переход $\delta'(p, a) = q$.

Шаг 3. Если при выполнении шага 2 сформировано новое, неисследованное состояние ДКА, повторим для него шаг 2.

Шаг 4. Состояние ДКА, в состав которого входит любое из допускающих состояний НКА, помечаем как допускающее состояние ДКА. ●

Пример 2.1. Построение ДКА по НКА.

Приведем к детерминированному виду НКА М2.

Шаг 1. Начальное состояние ДКА равно $[A]$.

Шаг 2. $\delta(A, a) = \{A, B\}$. Формируем состояние $[AB]$, записываем переход $\delta'([A], a) = [AB]$. $\delta(A, b) = \{A\}$, записываем переход $\delta'([A], b) = [A]$.

Шаг 3. Обнаружено новое состояние $[AB]$.

Шаг 2. $\delta(A, a) = \{A, B\}$, $\delta(B, a) = \varnothing$.

Записываем переход $\delta'([AB], a) = [AB]$. $\delta(A, b) = \{A\}$, $\delta(B, b) = \{S\}$.

Формируем состояние $[AS]$, записываем переход $\delta'([AB], b) = [AS]$.

Шаг 3. Обнаружено новое состояние $[AS]$.

Шаг 2. $\delta(A, a) = \{A, B\}$, $\delta(S, a) = \varnothing$.

Записываем переход $\delta'([AS], a) = [AB]$. $\delta(A, b) = \{A\}$, $\delta(S, b) = \varnothing$.

Записываем переход $\delta'([AS], b) = [A]$.

Шаг 3. Новых состояний не обнаружено.

Шаг 4. $F' = \{[AS]\}$, т.к. $S \in F$. ●

2.6. Конечные автоматы и регулярные выражения

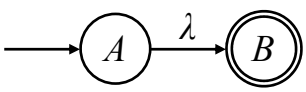
2.6.1. От регулярного выражения к конечному автомату

Метод построения конечного автомата по регулярному выражению заключается в построении частичных автоматов, соответствующих операндам выражения, и комбинации этих автоматов в порядке выполнения регулярных операций. Следующие правила рекурсивно строят частичные автоматы и автомат в целом.

1. Выражению 0 соответствует конечный автомат:

Никакого пути между начальным и финальным состояниями нет.

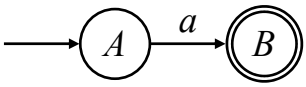
2. Выражению 1 соответствует конечный автомат:



Здесь используется так называемый *пустой* переход, или λ-переход.

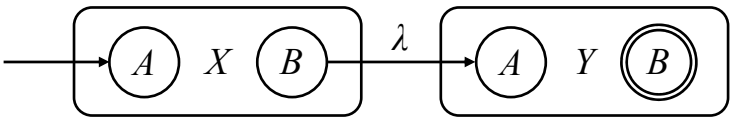
Пустой переход выполняется спонтанно, никакое событие для этого не требуется, при этом считывающая головка не перемещается.

3. Выражению a, $a \in \Sigma$, соответствует конечный автомат:



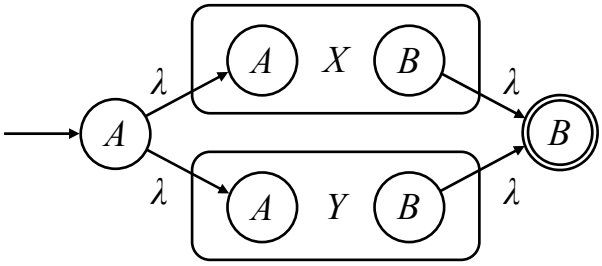
Автомат содержит единственный переход.

4. Выражению XY, где X и Y — регулярные выражения, для которых определены частичные автоматы M_X и M_Y, соответствует автомат:

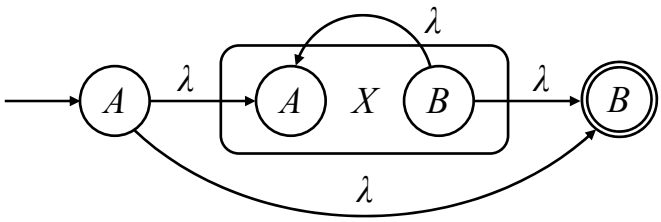


Частичные автоматы M_X и M_Y могут соединяться также совмещением финального состояния M_X с начальным состоянием M_Y.

5. Выражению $X + Y$, где X и Y — регулярные выражения, для которых определены частичные автоматы M_X и M_Y, соответствует автомат:



6. Выражению X^*, где X — регулярное выражение, для которого определен частичный автомат M_X, соответствует автомат:



Пустой переход из финального состояния M_X в начальное состояние M_X соответствует итерации выражения X^*. Пустой переход из начального состояния нового автомата в его финальное состояние соответствует пустой цепочке выражения X^*.

7. Выражению (X) соответствует конечный автомат выражения X. ●

Результирующий автомат называется *λ-НКА* (*НКА* с λ-переходами, иначе называемый *ε-НКА*, *e-НКА*). Этот автомат далее детерминируют (и минимизируют). *λ-НКА* является наиболее общим случаем конечного автомата, эквивалентным НКА и ДКА в смысле распознавания именно регулярных языков.

Пример. Построение автомата по регулярному выражению.

В качестве примера построим конечный автомат для регулярного выражения $r = (a+b)^*ab$.

Сначала нужно выстроить дерево выражения r, которое позволит определить частичные автоматы, которые должны быть построены. Дерево регулярного выражения строится аналогично дереву обычного арифметического выражения (рисунок 2.12).

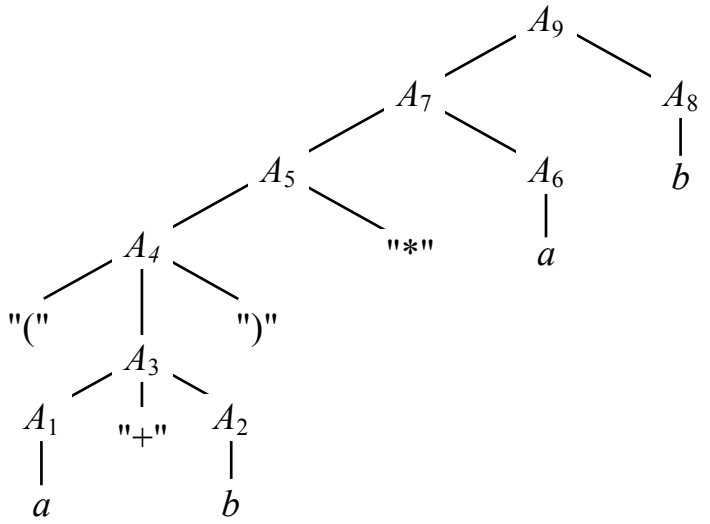


Рисунок 2.12. Дерево регулярного выражения $(a+b)^*ab$

Как видно из дерева выражения, для построения конечного автомата A_9 нужно построить 8 частичных автоматов, обозначенных на дереве A_1—A_8. Заметим, что автоматы A_3 и A_4 являются эквивалентными, тем не менее, без подробной детализации не обойтись.

Автоматы строятся в порядке их нумерации. Построение достаточно тривиально, результирующий автомат приведен на рисунке 2.13.

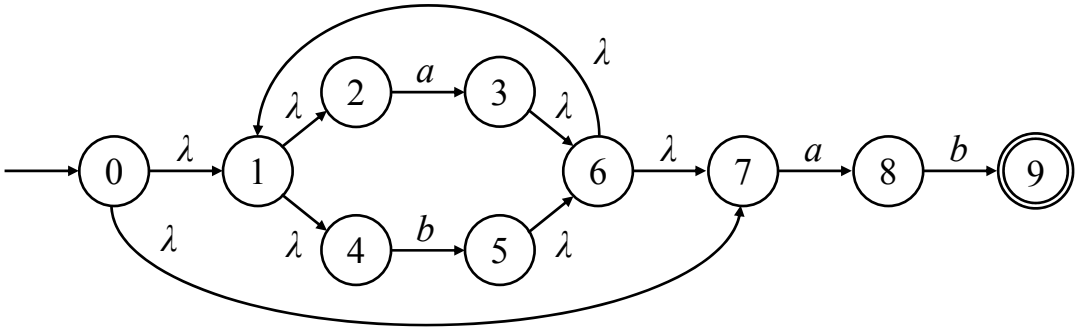


Рисунок 2.13. Результирующий λ-*НКА* A_9

Здесь для получения более простого автомата при последовательном соединении использовались не λ-переходы, а совмещение состояний. ●

2.6.2. Детерминизация *λ-НКА*

Автомат *λ-НКА* должен быть детерминирован (и минимизирован).

При этом возникает проблема обработки пустых переходов, которая решается при помощи так называемого *λ-замыкания* (*λ-closure*, а также *ε-closure*, *эпсилон-замыкание*).

Рассмотрим детерминизацию *λ-НКА* на примере автомата A_9.

Лямбда-замыкание некоторого состояния q — это множество всех состояний, достижимых из состояния q по пустым переходам, включая само состояние q.

Например, *λ*-замыкание состояния 0 автомата A_9 составляет множество состояний {0, 1, 2, 4, 7}. Обозначим это состояние как состояние *A* детерминированного автомата: $A = \lambda\text{-}closure(0)$.

Далее анализируются переходы из состояния *A* и получаются новые состояния ДКА:

- по символу *a* автомат переходит во множество состояний {3, 8}, замыкание которого {1, 2, 3, 4, 6, 7, 8}, обозначим состояние как *B*;
- по символу *b* автомат переходит в состояние 5, замыкание которого равно {1, 2, 4, 5, 6, 7}, обозначим состояние как *C*.

Из состояния B = {1, 2, 3, 4, 6, 7, 8}, автомат переходит:

- по символу *a* — в состояние 5, равное состоянию *B*;
- по символу *b* — во множество состояний {5, 9}, замыкание которого равно {1, 2, 4, 5, 6, 7, 9}, обозначим состояние как *D*.

Из состояния C = {1, 2, 4, 5, 6, 7}, автомат переходит:

- по символу *a* в состояние *B*;
- по символу *b* в состояние *C*.

Наконец, из состояния D = {1, 2, 4, 5, 6, 7, 9} автомат переходит:

- по символу *a* — состояние *B*;
- по символу *b* — в состояние *C*.

Результирующий ДКА приведен на рисунке 2.14, обозначим его М3.

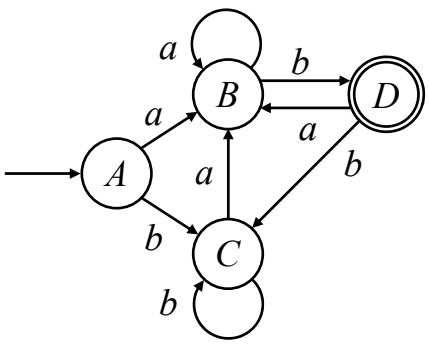


Рисунок 2.14. ДКА М3

Этот автомат является избыточным (не минимально возможным).

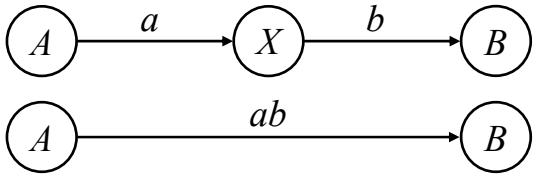
2.6.3. От конечного автомата к регулярному выражению

Переход от конечного автомата к регулярному выражению проще всего выполнить при помощи метода исключения состояний.

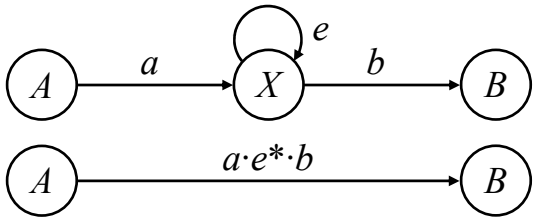
Суть метода состоит в том, чтобы заменить внутренние состояния автомата соответствующими регулярными выражениями. После замены между начальным и допускающим состоянием останется регулярное выражение, соответствующее автомату. Эта простая идея имеет ряд особенностей. Рассмотрим несколько примеров.

На следующем рисунке показан самый простой случай исключения, исключается состояние *X*. Проходя по стрелке от *A* к *X*, записываем метку стрелки *a*. Проходя от *X* к *C*, записываем метку *b*.

Тогда результирующее выражение — это *ab*, конкатенация *a* и *b*:

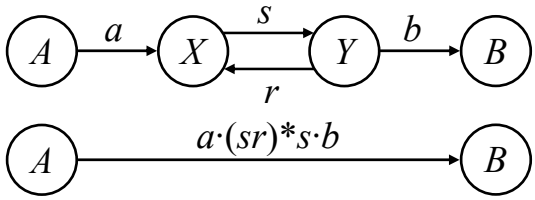


Более сложный случай возникает, если из состояния *X* есть стрелка в состояние *X* (цикл). Если стрелка помечена одним символом, например, *e*, то состояние *X* вставляет в поток конкатенации итерацию *e**:



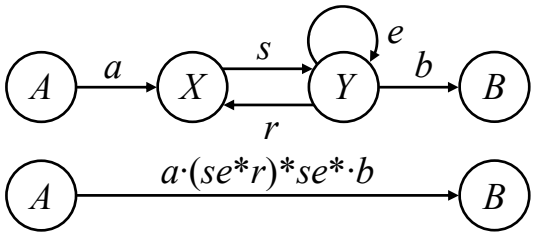
Если стрелка помечена несколькими символами, они соединяются операцией объединения. Например, если стрелка помечена символами *e* и *j*, итерация записывается как (*e*+*j*)*.

Итерация может быть неявной, когда между двумя состояниями есть стрелки в двух направлениях, например:



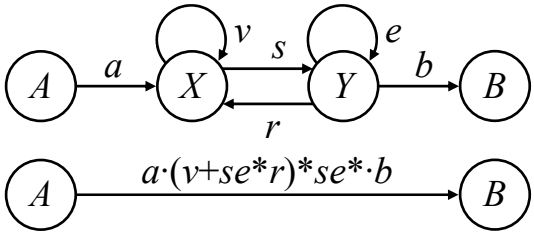
Прямой проход от *a* к *b* равен *s*. Из *X* в *X* можно пройти путем (*sr*)*, в целом получаем (*sr*)**s*. Выражение (*sr*)**s* может быть записано и как *s*(*rs*)*, если за точку отсчета вместо *X* принять *Y*.

Несколько сложнее ситуация, когда неявная итерация сочетается с явной в одном из состояний, например в состоянии *Y*:



Примем за отправную точку состояние *X*. Тогда из *X* к *b* прямой путь равен *se**. Из *X* в *X* можно пройти путем *se*r*, который не является обязательным. Окончательно получим (*se*r*)**se**.

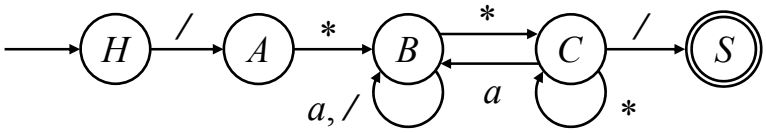
Теперь рассмотрим случай, когда есть и неявная итерация, и явная итерация как в состоянии *Y*, так и в состоянии *X*:



Примем за отправную точку состояние *X*. Из *X* к *b* прямой путь *se**. Из *X* в *X* можно пройти путями *v** или *se*r*, которые не обязательны.

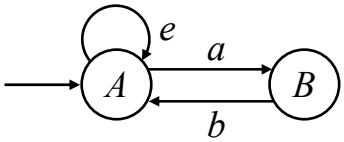
Таким образом, требуемое выражение (*v*+*se*r*)**se**.

Теперь мы можем легко вычислить регулярное выражение для автомата M1, распознающего комментарий языка *C*:



Просто подставим другие знаки и получим: /*·(*a* +/ + ****a*)****·/.

Если есть переходы в начальное или из допускающих состояний, то эти состояния тоже учитываются, например:



Для данного фрагмента регулярным выражением будет (*e*+*ab*)**a*.

В сложных случаях можно вставить пустой переход от фиктивного состояния к начальному, поскольку пустой переход не изменяет выражение. Аналогично можно поступить и с допускающим состоянием.

2.7. Конечные автоматы и регулярные грамматики

2.7.1. От конечного автомата к праволинейной грамматике

Существует класс грамматик, а именно праволинейные автоматные грамматики, которые порождают цепочки, распознаваемые конечными автоматами. Конструирование праволинейной автоматной грамматики по конечному автомату описывает следующий алгоритм.

Алгоритм 2.2. Построение грамматики по конечному автомату.

На входе: конечный автомат $M(Q, \Sigma, \delta, q_0, \{f\})$.

На выходе: автоматная грамматика $G(\Sigma, N, P, A_0)$.

Шаг 1. $N = Q — F$, $P = \{\}$, $S = q_0$.

Шаг 2. Рассматриваем переходы конечного автомата $B \in \delta(A, a)$.

а) записываем в P правило $A \to aB$, если $B \notin F$.

б) записываем в P правило $A \to a$, если $B \in F$.

в) записываем в P правило $A \to aB$, если $B \in F$, и существует переход из B: $C \in \delta(B, b)$; записываем нетерминал B в N. ●

В примерах далее используется конечный автомат M4:

M4 = ({*A*, *B*, *E*, *F*}, {*a*, *b*, *c*, *d*}, δ, {*A*}, {*E*, *F*}), δ = {

δ(*A*, *a*) = *B*,

δ(*B*, *b*) = *E*,

δ(*E*, *c*) = *B*,

δ(*E*, *d*) = *F*

}.

Граф переходов автомата M4 приведен на следующем рисунке:

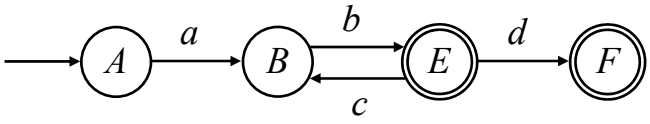


Рисунок 2.15. ДКА M4

Пример 2.2. Построение грамматики по конечному автомату.

Построим праволинейную грамматику для автомата M4.

Шаг 1. $\Sigma = \{a, b, c, d\}$, $N = \{A, B\}$, $P = \{\}$, $S = A$.

Шаг 2. а) по переходу $\delta(A, a) = B$ запишем правило $A \to a$.

б) по переходу $\delta(B, b) = E$ запишем правило $B \to b$ и $B \to bE$, $B \in N$.

в) по переходу $\delta(B, b) = E$ запишем правило $B \to bE$, запишем E в N.

а) по переходу $\delta(E, c) = B$ запишем правило $E \to cB$.

б) по переходу $\delta(E, d) = F$ запишем правило $E \to d$.

Результирующая праволинейная автоматная грамматика:

$A \to aB$
$B \to b \mid bE$
$E \to cB \mid d$ ●

2.7.2. От праволинейной грамматики к конечному автомату

Построить конечный автомат по праволинейной автоматной грамматике поможет следующий алгоритм.

Алгоритм 2.3. Построение автомата по праволинейной грамматике.
На входе: праволинейная автоматная грамматика G = (Σ, N, P, *S*).
На выходе: конечный автомат *M* = (Q, Σ, δ, q_0, F).
Шаг 1. Q = N, $q_0 = S$, F = {}, δ = {}.
Шаг 2. Просматриваем правила грамматики.
а) если для правила вида $A \to aB$ нет пары $A \to a$, то
записываем переход $B \in \delta(A, a)$.
б) если для правила $A \to a$ есть пара $A \to aB$, то записываем переход
$B \in \delta(A, a)$, F = F∪{*B*}.
в) если для правила $A \to a$ нет пары $A \to aB$, то записываем переход
$f \in \delta(A, a)$, F = F∪{*f*}, Q = Q∪{*f*}, *f* — новое состояние. ●

Рассмотрим пример преобразования грамматики в автомат.
Пример 2.3. Построение автомата по праволинейной грамматике.
Построим конечный автомат на основе праволинейной грамматики:

$A \to aB$
$B \to b \mid bE$
$E \to cB \mid d$

Шаг 1. Σ = {*a*, *b*, *c*, *d*}; Q = {*A*, *B*, *E*}, $q_0 = A$, F = {}, δ = {}.
Шаг 2. Просматриваем правила грамматики.
а) из правила $A \to aB$ следует $B \in \delta(A, a)$;
б) из правил $B \to b \mid bE$ следует $E \in \delta(B, b)$, F = F∪{*E*};
а) из правила $E \to cB$ следует $B \in \delta(E, c)$;
в) из правила $E \to d$ следует $C \in \delta(E, d)$, F = F∪{*C*};

Полученный конечный автомат является автоматом M4:

M = ({*A*, *B*, *E*, *C*}, {*a*, *b*, *c*, *d*}, δ, {*A*}, {*E*, *C*}), δ = {
δ(*A*, *a*) = {*B*},
δ(*B*, *b*) = {*E*},
δ(*E*, *c*) = {*B*},
δ(*E*, *d*) = {*C*}
} ●

2.8. Регулярные грамматики и выражения

Автоматная грамматика может быть легко преобразована не только в конечный автомат, но и в регулярное выражение. Немногим более сложным является и переход от регулярного выражения к регулярной грамматике.

Для перехода от автоматной грамматики к регулярному выражению на основе правил грамматики строится система уравнений с регулярными коэффициентами, решение которой дает искомое выражение.

Для перехода от регулярного выражения к регулярной грамматике используется метод, напоминающий построение конечного автомата по регулярному выражению.

2.8.1. Системы уравнений с регулярными коэффициентами

Простейшие уравнения с регулярными коэффициентами в алфавите Σ имеют следующий вид:

$$X = \alpha X + \beta , \qquad (2.1)$$

$$X = X\alpha + \beta . \qquad (2.2)$$

Здесь $\alpha, \beta \in \Sigma^*$ — регулярные выражения, $X \notin \Sigma$ — переменная.

Первая запись является правосторонней, вторая — левосторонней. Эти записи не равнозначны (операция конкатенации не коммутативна).

Решением первого уравнения (2.1) является регулярное множество, обозначаемое регулярным выражением $\alpha^*\beta$. Решением второго уравнения (2.2) является регулярное множество, обозначаемое регулярным выражением $\beta\alpha^*$. Доказано, что они являются наименьшими из возможных решений для данных уравнений; их называют *наименьшей неподвижной точкой*.

Уравнения вида (2.1) используются при переходе от праволинейной грамматики к регулярному выражению. Соответственно, уравнения вида (2.2) используются при построении регулярного выражения по леволинейной грамматике. В связи с этим различают лево- и правосторонние системы уравнений с регулярными коэффициентами.

Система уравнений в правосторонней записи имеет следующий вид:

$$X_1 = \alpha_{10} + \alpha_{11}X_1 + \alpha_{12}X_2 + \ldots + \alpha_{1n}X_n$$
$$X_2 = \alpha_{20} + \alpha_{21}X_2 + \alpha_{22}X_2 + \ldots + \alpha_{2n}X_n$$
$$\ldots$$
$$X_n = \alpha_{n0} + \alpha_{n1}X_1 + \alpha_{n2}X_2 + \ldots + \alpha_{nn}X_n$$

Левосторонняя система уравнений отличается от правосторонней только расположением переменных X_i не справа, а слева от регулярных коэффициентов α_{ij}.

Для построения правосторонней системы уравнений с регулярными коэффициентами есть следующий формальный алгоритм.

Алгоритм 2.4. Построение системы уравнений с регулярными коэффициентами по праволинейной грамматике.

На входе: правосторонняя автоматная грамматика G = (Σ, N, P, *S*).

На выходе: регулярное выражение *r* над алфавитом Σ.

Шаг 1. Обозначить нетерминальные символы $X_1, X_2, \ldots, X_n$.

Шаг 2. Определить регулярные коэффициенты по правилам:

а) если $(X_i \rightarrow \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_m) \in P$, то $\alpha_{i0} = (\gamma_1 + \gamma_2 + \ldots + \gamma_m)$.

б) если $(X_i \rightarrow \gamma_1 X_j \mid \gamma_2 X_j \mid \ldots \mid \gamma_m X_j) \in P$, то $\alpha_{ij} = (\gamma_1 + \gamma_2 + \ldots + \gamma_m)$.

в) принять все другие коэффициенты равными нулю.

Шаг 3. Построить систему уравнений с переменными $X_1, X_2, \ldots, X_n$ и полученными регулярными коэффициентами α_{ij}. ●

Этот алгоритм используется при автоматизации процесса. Для ручных преобразований построить систему уравнений по грамматике можно просто. Для примера построим систему уравнений по правосторонней грамматике, эквивалентной автомату M1 (рисунок 2.8):

$$H \rightarrow /A \qquad (2.3)$$
$$A \rightarrow *B$$
$$B \rightarrow aB \mid /B \mid *C$$
$$C \rightarrow *C \mid aB \mid /$$

Переход к системе уравнений выполняется так: стрелки → заменяем знаком "=", а вертикальные черточки заменяем знаком "+".

$$H = /A \qquad (2.4)$$
$$A = *B$$
$$B = aB + /B + *C$$
$$C = *C + aB + /$$

После этого нужно сгруппировать нетерминалы в третьей строке, и сформировать соответствующий коэффициент α:

$$B = (a + /)B + *C \quad (2.5)$$

Далее построенную систему уравнений нужно решить, и получить выражение для целевого символа.

Решение использует метод последовательных подстановок. Смысл метода в том, чтобы находить решения для уравнений и подставлять эти решения в другие уравнения до тех пор, пока все уравнения не будут решены, или до тех пор, пока не будет решено уравнение для целевого символа. При этом делается два прохода по уравнениям, сверху вниз и обратно. Это напоминает решение системы линейных алгебраических уравнений методом Гаусса.

Алгоритм 2.5. Решение системы уравнений с регулярными коэффициентами.

На входе: правосторонняя система уравнений с регулярными коэффициентами. На выходе: регулярные выражения для переменных.

Шаг 1. Положить $i = 1$.

Шаг 2. Если $i = n$, то перейти к шагу 4.

Записать i-тое уравнение в виде $X_i = \alpha_i X_i + \beta_i$.

В уравнения с переменными $X_{i+1}, \ldots, X_n$ подставить $\alpha_i * \beta_i$ вместо X_i.

Шаг 3. Положить $i = i+1$. Перейти к шагу 2.

Шаг 4. Записать уравнение для X_n в виде $X_n = \alpha_n X_n + \beta_n$.

Шаг 5. Решить уравнение для X_i, $X_i = \alpha_i * \beta_i$.

Подставить $\alpha_i * \beta_i$ вместо X_i в уравнения для переменных $X_{i-1}, \ldots, X_1$.

Шаг 6. Если $i = 1$, то конец алгоритма.

Положить $i = i-1$, перейти к шагу 5. ●

2.8.2. От грамматики к регулярному выражению

Пример. От праволинейной грамматики к регулярному выражению

Построим регулярное выражение на основе грамматики (2.3), описывающей комментарий языка C. Система уравнений с регулярными коэффициентами приведена в (2.4) с учетом поправки (2.5).

Будем решать систему методом подстановок интуитивно. Ближе всего к допускающему состоянию S нетерминал C, с него и начинаем.

Уравнение для C нужно привести к виду $C = \alpha C + \beta$. По случаю, оно в таком виде и есть. Получаем $\alpha = *$, $\beta = aB + /$.

Решением уравнения является $\alpha^*\beta$:

$C = **(aB + /)$.

Подставим решение для C в уравнение для B, получим:

$B = (a + /)B + ***(aB + /)$.

Раскрываем правые скобки для объединения коэффициентов при B:

$B = (a + /)B + ***aB + ***/$.

Объединяем коэффициенты при B:

$B = (a + / + ***a)B + ***/$.

Находим $\alpha = (a + / + ***a)$, $\beta = ***/$, и решение уравнения для B:

$B = (a + / + ***a)****/$.

Подставляем решение для B сначала в A, затем в H, после чего сразу получаем окончательное решение:

$H = /*(a + / + ***a)****/$.

Второй проход не понадобился, решение для C нас не интересует. ●

2.9. Минимизация конечных автоматов

Можно доказать, что существует ДКА такой, что он имеет наименьшее число состояний из всех ДКА, эквивалентных ему с точностью до обозначения состояний. Минимизация имеет значение по той причине, что, во-первых, минимальный автомат можно более просто реализовать программно, а во-вторых, при преобразовании регулярного выражения в автомат обычно получается НКА с эквивалентными состояниями.

2.9.1. Эквивалентность состояний

Вычисление эквивалентных состояний позволяет дать ответы на следующие два вопроса:

1) если есть два регулярных выражения или два конечных автомата, или выражение и автомат, то можно ли доказать их эквивалентность?

2) если есть некоторый конечный автомат, то можно ли построить эквивалентный автомат с меньшим числом состояний?

Говорят, что два состояния p и q *эквивалентны*, если любая входная цепочка w переводит автомат из состояния p в допускающее состояние тогда и только тогда, когда эта цепочка переводит автомат из состояния q в допускающее состояние.

Если же есть такая цепочка w, которая переводит автомат из одного из состояний p и q в допускающее состояние, а из другого состояния в не допускающее состояние, то говорят, что состояния p и q *различимы*.

Эквивалентные состояния можно вычислить при помощи алгоритма заполнения таблицы, который выстраивает таблицу *неэквивалентности* состояний некоторого СДКА $M = (Q, \Sigma, \delta, q_0, F)$. Эта таблица содержит отметку в ячейке (p, q), если пара состояний p и q различимы.

Начальными различимыми состояниями являются множества допускающих и не допускающих состояний: F и Q — F.

Алгоритм заполнения таблицы неэквивалентности

Базис. Если $p \in F$ и $q \notin F$, то пара состояний (p, q) различима.

Индукция. Если r и s — различимые состояния, и существуют переходы в них $\delta(p, a) = r$ и $\delta(q, a) = s$, то состояния p и q также различимы.

Доказательство: если r и s различимы, то должна существовать цепочка w, для которой $(r, w) \vdash^* (f_1, x_1)$ и $(s, w) \vdash^* (f_2, x_2)$, и только одно из состояний f_1 и f_2 является допускающим. Если есть пара состояний (p, q) таких, что $\delta(p, a) = r$ и $\delta(q, a) = s$, то цепочка aw различает p и q, так как $(p, aw) \vdash^* (f_1, x_1)$ и $(q, aw) \vdash^* (f_2, x_2)$. ●

Заметим, что минимизируемый автомат должен быть полным ДКА, иначе нужно ввести крайнее состояние и добавить переходы в него. В качестве примера будем рассматривать СДКА M5 (рисунок 2.16) [12].

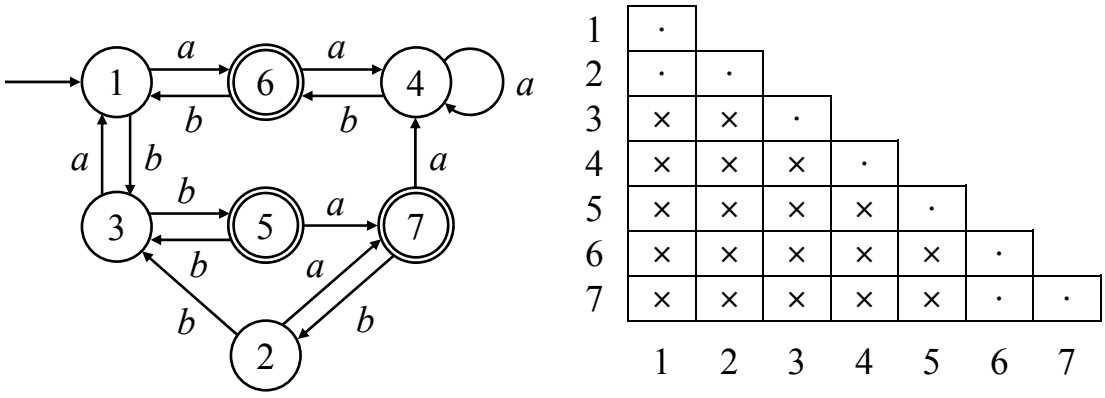


Рисунок 2.16. Таблица неэквивалентности состояний СДКА M5

Сначала пары состояний, в которых одно является допускающим, а другое нет, помечаются в таблице как различимые по базису алгоритма. Для рассматриваемого примера это пары 5-*x*, 6-*x* и 7-*x*, где *x* — любое состояние, кроме 5, 6 и 7. При этом состояния 5, 6 и 7 не помечены.

Теперь рассмотрим все возможные пары, для которых в таблице нет отметок, это пары 2-1, 3-1, 3-2, 4-1, 4-2, 4-3, 5-6, 5-7, 6-7.

Рассматриваем переходы этих пар.

3-1: $\delta(3, a) = 1$, $\delta(1, a) = 6$, пара 1-6 помечена, помечаем пару 3-1.

3-2: $\delta(3, a) = 1$, $\delta(2, a) = 7$, пара 1-7 помечена, помечаем пару 3-2.

4-1: $\delta(4, a) = 4$, $\delta(1, a) = 6$, пара 4-6 помечена, помечаем пару 4-1.

4-2: $\delta(4, a) = 4$, $\delta(2, a) = 7$, пара 4-7 помечена, помечаем пару 4-2.

4-3: $\delta(4, a) = 4$, $\delta(3, a) = 1$, пара 4-1 помечена, помечаем пару 4-3.

5-6: $\delta(5, a) = 7$, $\delta(6, a) = 4$, пара 7-4 помечена, помечаем пару 5-6.

5-7: $\delta(5, a) = 7$, $\delta(7, a) = 4$, пара 7-4 помечена, помечаем пару 5-7.

Другие пары помечены быть не могут, пары 2-1 и 6-7 остались не помеченными, следовательно, они эквивалентны.

2.9.2. Минимизация методом Хопкрофта

Минимизации заключается в объединении эквивалентных состояний с соответствующей поправкой функции переходов. В основе алгоритма Хопкрофта лежит определение классов эквивалентности, которые равны состояниям минимального автомата.

Два состояния p и q находятся в одном классе эквивалентности тогда и только тогда, когда они эквивалентны. Это означает, что два эквивалентных состояния не могут находиться в разных классах; с другой стороны, два неэквивалентных состояния находятся каждый в своем классе.

Все состояния исходного автомата группируются в блоки, которые содержат только эквивалентные состояния.

Состояние p, не имеющее эквивалентного, формирует блок {p} из одного состояния. Пара эквивалентных состояний (p, q) либо входит в имеющийся блок, содержащий либо p, либо q, либо формирует новый блок {p, q}.

Если блок содержит более одного состояния, то они объединяются и функция переходов минимального автомата δ' корректируется следующим образом, (p и q — эквивалентные состояния, r — любое другое):

1) $\delta'(\{p, q\}, a) = \delta(p, a) \cup \delta(q, a), \forall a \in \Sigma$;

2) если $\delta(r, a) = p$ или $\delta(r, a) = q$, то $\delta'(r, a) = \{p, q\}$.

Схема минимизации по Хопкрофту

На входе: СДКА $M = (Q, \Sigma, \delta, q_0, F)$.

На выходе: ДКА $M' = (Q', \Sigma, \delta', q_0', F')$.

1. Вычислить таблицу неэквивалентности.

2. Определить блоки эквивалентных состояний.

3. Состояния, входящие в один блок, объединить.

4. Начальным состоянием M' является блок, содержащий q0.

5. Множество допускающих состояний M' включает блоки, в которые входит хотя бы одно допускающее состояние M. ●

Пример минимизации по Хопкрофту

Минимизируем СДКА M5.

1. Таблица неэквивалентности приведена на рисунке 2.16.

2. Блоки эквивалентных состояний: {12}, {3}, {4}, {5} и {67}.

3. $\delta'(\{12\}, a) = \{67\}$, т.к. $\delta(1, a) = 6$ и $\delta(2, a) = 7$;

$\delta'(\{12\}, b) = \{3\}$, т.к. $\delta(1, b) = 3$ и $\delta(2, b) = 3$;

$\delta'(\{67\}, a) = \{4\}$, т.к. $\delta(6, a) = 4$ и $\delta(7, a) = 4$;

$\delta'(\{67\}, b) = \{12\}$, т.к. $\delta(6, b) = 1$ и $\delta(7, b) = 2$;

$\delta'(\{3\}, a) = \{12\}$, т.к. $\delta(3, a) = 1$;

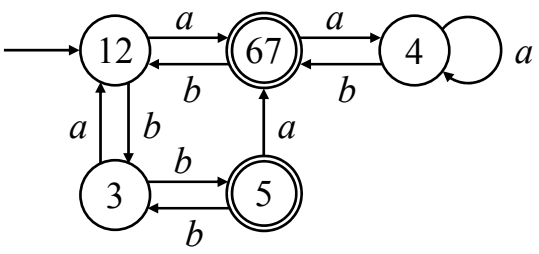
$\delta'(\{4\}, b) = \{67\}$, т.к. $\delta(4, b) = 6$;

$\delta'(\{5\}, a) = \{67\}$, т.к. $\delta(5, a) = 7$;

4. Начальное состояние {12}.

5. Допускающие состояния {5} и {67}.

Граф переходов минимального автомата изображен на следующем рисунке:



●

2.9.3. Языки состояний конечного автомата

Пусть есть некоторый конечный автомат $M(Q, \Sigma, \delta, I, F)$.

Левым языком $L(q)$ состояния q автомата M будем называть язык, допускаемый конечным автоматом $M_L(Q, \Sigma, \delta, I, \{q\})$. Автомат M_L образован из исходного заменой множества *финальных* состояний F единственным состоянием q. Иначе говоря, левый язык состояния q — это множество цепочек, которые ведут из множества начальных состояний в интересующее состояние q.

Правым языком $R(q)$ состояния q автомата M будем называть язык, допускаемый конечным автоматом $M_R(Q, \Sigma, \delta, \{q\}, F)$. Автомат M_R образован из исходного заменой множества *начальных* состояний I единственным состоянием q. Иначе говоря, правый язык состояния q — это множество цепочек, которые ведут из интересующего состояния q в любое из финальных состояний.

Должны выполняться следующие очевидные равенства:

$L(F) = R(I) = L(M)$,

$\bigcup L(q) \cdot R(q) = L(M)$, $\forall q \notin (I \cup F)$.

Следующий автомат, назовем его A, будем использовать как пример:

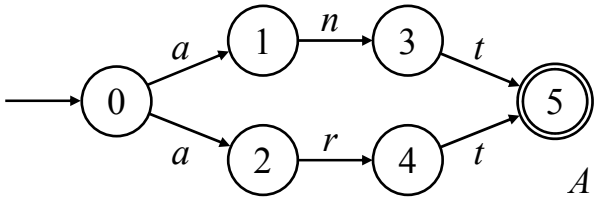


Рисунок 2.17. Автомат A

Заметим, что наличие двух и более одинаково помеченных стрелок из одного состояния указывает на то, что автомат не детерминирован.

Наличие таких же стрелок, ведущих в одно и то же состояние, указывает на то, что автомат, *возможно*, избыточен (не минимален).

Утверждение 2.1. Конечный автомат является детерминированным *тогда и только тогда*, когда левые языки всех его состояний попарно не пересекаются (не имеют общих одинаковых цепочек).

Утверждение 2.2. Конечный автомат является минимальным *тогда и только тогда*, когда правые языки всех его состояний различны.

Автомат A недетерминирован и избыточен, так как для него не выполняется ни одно из утверждений. В этом можно убедиться, сравнивая левые и правые языки:

$L(0) = 0$, $L(1) = a$, $L(2) = a$, $L(3) = an$, $L(4) = ar$, $L(5) = ant + art$.

$R(0) = ant + art$, $R(1) = nt$, $R(2) = rt$, $R(3) = t$, $R(4) = t$, $R(5) = 0$.

Очевидно, что $L(1) \cap L(2) = a$, и $R(3) = R(4) = t$.

2.9.4. Обращенный конечный автомат

Пусть задан конечный автомат $M(Q, \Sigma, \delta, I, F)$, назовем его *прямым*.

Инвертированный, или *обращенный* конечный автомат — это автомат $M^R(Q, \Sigma, \delta^R, F, I)$, функция переходов которого определяется так: если в δ есть переход $r \in \delta(q, a)$, в δ^R записывается переход $q \in \delta(r, a)$.

Иначе говоря, начальные и финальные состояния инвертированного автомата меняются местами, а направления переходов (стрелок на графе переходов) меняются на противоположные.

На рисунке 2.18 показан обращенный конечный автомат *A*.

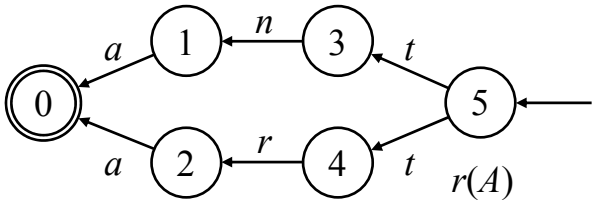


Рисунок 2.18. Обращенный автомат *A*

После инвертирования признаки не детерминированного и не минимизированного конечного автомата меняются на противоположные.

Стрелки 0→1 и 0→2 прямого автомата указывали на недетерминированность, а в обращенном автомате они указывают на избыточность.

Стрелки 3→5 и 4→5 прямого автомата указывали на избыточность, а в обращенном автомате они указывают на недетерминированность.

Говоря образно, недетерминированность — это противоположность, или обратная сторона избыточности.

При обращении происходит также взаимная замена левых и правых языков: обращенный левый язык некоторого состояния q прямого автомата становится правым языком этого состояния обращенного автомата и наоборот, обращенный правый язык состояния q прямого автомата становится левым языком этого состояния в обращенном автомате.

Обозначим язык *L* обращенного автомата как L_R, обращенный язык *L* как L^R (в обращенном языке цепочки записаны в обратном порядке).

Тогда:

$R_R(q) = L^R(q)$, $L_R(q) = R^R(q)$, и $L_R(M) = L^R(M)$.

2.9.5. Минимизация по методу Бржозовского

Существует еще один оригинальный метод минимизации конечных автоматов — алгоритм Бржозовского (1962). В отличие от других алгоритмов, он позволяет минимизировать недетерминированные конечные автоматы.

Алгоритм Бржозовского

Алгоритм Бржозовского использует следующие две операции над конечными автоматами, определяемые как функции (операции):

- *детерминизация* конечного автомата $d(A)$;
- *обращение* конечного автомата (*реверс*, *reverse*) $r(A)$.

Минимизация по методу Бржозовского сводится к дважды выполняемой последовательности операций обращения и детерминизации:

$A_{min} = d(r(d(r(A))))$, или, сокращенно, $A_{min} = drdr(A)$.

Рассмотрим работу алгоритма на примере автомата A (рисунок 2.19).

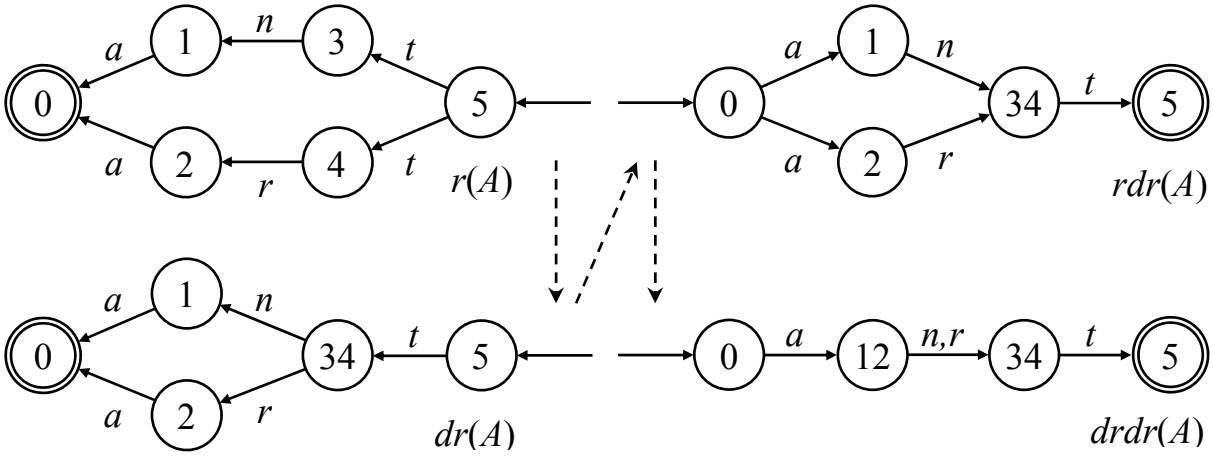


Рисунок 2.19. Минимизация автомата A

Первая пара операций $dr(A)$ устраняет избыточность.

В процессе первой детерминизации обращенного автомата $r(A)$ происходит объединение одинаковых левых языков, которые являются правыми языками прямого автомата A. Левые языки $L(3)$ и $L(4)$ при соединении состояний 3 и 4 объединились в левый язык $L(34) = t$.

Вторая пара операций $dr(dr(A))$ устраняет недетерминированность.

После второго обращения левые языки автомата $dr(A)$ становятся правыми языками автомата $rdr(A)$, которые были соединены в процессе детерминизации, поэтому, в силу утверждения 2.2, автомат $rdr(A)$ является минимальным.

Оставшаяся операция детерминизации устраняет недетерминированность прямого (или обращенного) автомата, объединяя одинаковые левые языки $L(1)$ и $L(2)$ прямого автомата соединяя состояния 1 и 2.

Результирующий автомат является детерминированным в силу конструкции формулы (последняя операция — детерминизация).

Автомат $drdr(A)$ распознает тот же язык, что и автомат A.

Это следует из того, что язык обращенного автомата является обращенным языком прямого, Так как обращение выполняется дважды, язык $drdr(A)$ равен языку A.

Разъединение соединения

Несмотря на кажущуюся простоту, алгоритм Бржозовского немного загадочнее. Рассмотрим минимальный не строгий ДКА, граф переходов которого приведен на рисунке 2.20.

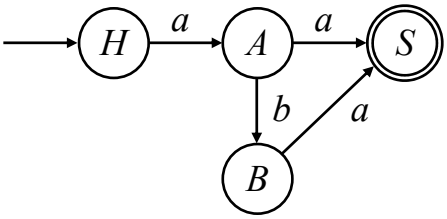


Рисунок 2.20. Автомат *Б*

По формальным признакам этот автомат кажется избыточным, — в состояние *S* ведут две стрелки, помеченные символом *a*. На рисунке 2.21 показаны графы автоматов, получающихся в процессе минимизации.

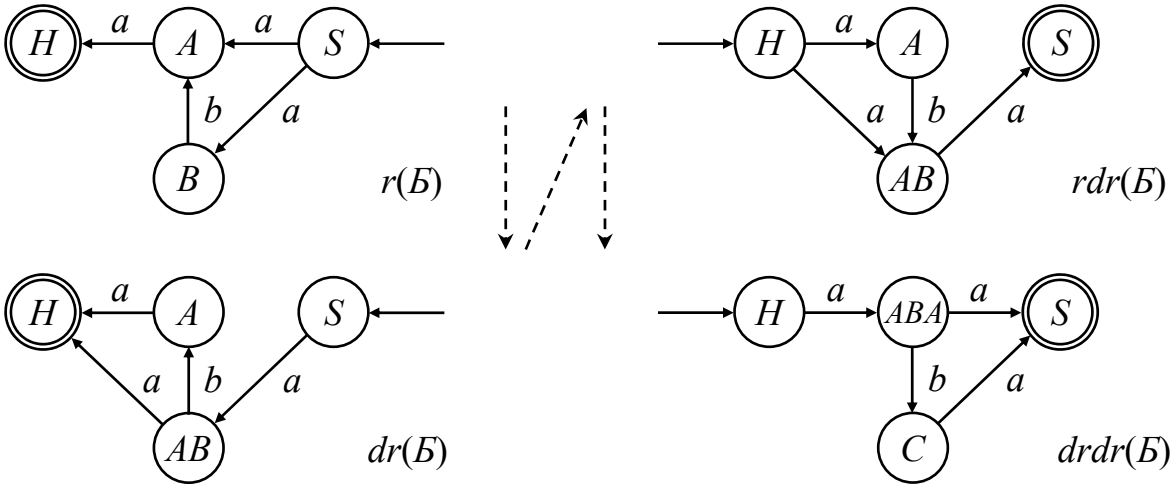


Рисунок 2.21. Минимизация автомата *Б*

Детерминизация *dr*(*Б*) объединяет *B* и *A*, убирая стрелку *S*→*A*, но добавляет стрелку *AB*→*H*, чтобы восстановить язык. При этом автомат *rdr*(*Б*) становится недетерминированным, что указывает на соединение неравных правых языков исходного автомата. Вторая детерминизация *drdr*(*Б*) соединяет *A* и *AB*, убирая стрелку *H*→*AB*, но добавляет стрелку *A*→*S*, восстанавливая язык. Произошло разъединение соединенных правых языков, и результирующий автомат эквивалентен исходному.

В этом заключается мистичность этого уникального алгоритма.

Алгоритм Бржозовского формирует неполный ДКА из СДКА, если алгоритм детерминизации строит только достижимые состояния.

Недостаток алгоритма Бржозовского лежит вне его самого, и определяется экспоненциальной сложностью алгоритма детерминизации.

2.10. Свойства регулярных языков

Свойства регулярных языков используются в следующих случаях.

1. Для доказательства нерегулярности некоторого языка.

2. Для построения новых регулярных языков, используя свойства их замкнутости относительно некоторых операций.

3. Для решения важных вопросов относительно автоматов и языков.

2.10.1. Лемма о разрастании для регулярных языков

Не каждый язык, который можно получить при помощи регулярных операций, будет регулярным. Лемма о разрастании, называемая также *леммой о накачке* (*pumping lemma*), позволяет доказать нерегулярность некоторого языка. Если для некоторого языка лемма не выполняется, то этот язык не является регулярным.

Неформально лемму можно описать следующим образом. Если дан регулярный язык и достаточно длинная цепочка символов этого языка, то в этой цепочке можно найти непустую подцепочку, которую можно повторить сколь угодно много раз, и все полученные таким способом новые цепочки будут принадлежать тому же регулярному языку.

Лемма о разрастании для регулярных языков.

Если L — регулярный язык, то существует константа $n > 0$ такая, что если $\alpha \in L$ и $|\alpha| \geq n$, то α можно записать в виде $\alpha = \delta\beta\gamma$, где $|\beta| \neq 0$, $|\delta\beta| \leq n$, и тогда $\alpha' = \delta\beta^i\gamma \in L \ \forall i \geq 0$. ●

Доказательство. Если L — регулярный язык, то ему соответствует минимальный конечный автомат M, имеющий n состояний. Возьмем цепочку α длиной не меньше n. Поскольку для распознавания n символов автомат должен выполнить n шагов, а при количестве состояний n может быть выполнено только n–1 шагов, то некоторые шаги автомата должны повторяться. Если $\alpha = \delta\beta\gamma$, то распознаванию δ соответствует $(q_0, \delta) \vdash^* (q_1, \lambda)$, и тогда распознаванию β соответствует $(q_1, \beta) \vdash^* (q_2, \lambda)$, после чего распознается цепочка γ, для которой $(q_2, \gamma) \vdash^* (r, \lambda)$, $r \in F$. Но тогда должно выполняться $(q_1, \beta^i) \vdash^* (q_2, \lambda), \forall i \geq 0$. ●

Заметим, что таким образом проверяются языки, цепочки которых могут быть сколь угодно длинными, при этом длина цепочки превышает количество состояний автомата, что возможно только тогда, когда язык (автомат) содержит итерацию. Эти языки являются «бесконечными».

Например, язык целых чисел без знака основан на итерации *цифра**, что позволяет уже в цепочке "00" сколь угодно раз повторять ноль. То же самое можно сказать о языке, описывающем идентификатор.

С другой стороны, некоторые регулярные языки конечны. Примером такого языка является простейший язык $a^2 = \{"aa"\}$. Лемма к конечным языкам неприменима, конечные языки регулярны по определению.

Одним из языков, использующих повторение, является язык, цепочки которого содержат сначала некоторое число символов a, за которыми следует такое же число символов b. Этот язык описывает следующая формула: $L = \{a^n b^n \mid n \geq 1\}$. Для языка L нельзя сформировать конечный автомат, потому что автомат запоминает только конкретное количество распознанных символов a, а не произвольно выбранное их число.

При помощи леммы о разрастании можно доказать, что язык L не является регулярным. Возьмем цепочку $\alpha = a^n b^n$, и представим ее в виде трех цепочек $\alpha = \delta\beta\gamma$. Тогда цепочка β может располагаться в одном из трех мест: в последовательности a^n, в последовательности b^n, и в точке соединения a^n и b^n. Если β находится в a^n или в b^n, то итерация β^0 нарушает равенство количеств a и b: либо число символов a будет меньше, либо число символов b. Если β находится в середине, начальная часть β содержит символы a, а конечная — символы b, и итерация β^2 нарушает следование символов b строго за символами a. Из этого следует, что L не является регулярным языком.

Если для того же языка L число n заранее известно, язык становится конечным и регулярным. Например, язык $L_2 = a^2 b^2$, для L_2 легко можно построить конечный автомат. С другой стороны, язык $L = a^n$, $n \geq 1$, тоже регулярный, хотя n заранее неизвестно.

Регулярным является также язык $L_{mn} = \{a^m b^n \mid m \geq 1, n \geq 1\}$. Для него можно найти цепочку β такую, что ее повторение произвольное число раз формирует новую цепочку, принадлежащую L_{mn}. Цепочка β выбирается либо в области a^m, либо в области b^n. Тот факт, что выбор цепочки β в точке соединения a^m и b^n нарушает чередование символов a и b, не говорит о нерегулярности, так как итерация найдена.

2.10.2. Свойства замкнутости

Множество называется замкнутым относительно операции θ, если результат выполнения операции θ над любыми элементами множества возвращает элемент, принадлежащий этому же множеству. Например, множество целых чисел замкнуто относительно операций сложения, умножения и вычитания, но не замкнуто относительно деления.

Пусть задан алфавит Σ и два регулярных языка: $L \subseteq \Sigma^*$ и $M \subseteq \Sigma^*$.

Пусть конечный автомат $(Q_1, \Sigma, \delta_1, \{q_1\}, \{f_1\})$ распознает язык L, а конечный автомат $(Q_2, \Sigma, \delta_2, \{q_2\}, \{f_2\})$ — язык M, и $Q_1 \cap Q_2 = \varnothing$.

Свойства замкнутости позволяют конструировать новые языки из регулярных языков L и M, которые также будут регулярными.

Регулярные языки замкнуты относительно следующих операций:

а) конкатенации:

язык $P = LM = \{ \alpha\beta \mid \alpha \in L, \beta \in M \}$ является регулярным;

доказательство: язык P распознается конечным автоматом

$(Q_1 \cup Q_2, \Sigma, \delta_1 \cup \delta_2 \cup \{ \delta(f_1, \lambda) = q_2 \}, \{q_1\}, \{f_2\})$;

б) объединения:

язык $P = L \cup M = \{ \alpha, \beta \mid \alpha \in L, \beta \in M \}$ является регулярным;

доказательство: язык P распознается конечным автоматом

$(Q_1 \cup Q_2, \Sigma, \delta_1 \cup \delta_2, \{q_1, q_2\}, \{f_1, f_2\})$;

в) итерации:

язык $P = L^n = \{ \alpha^n \mid \alpha \in L, n \geq 0 \}$ является регулярным;

доказательство: язык P распознается конечным автоматом

$(Q_1 \cup \{p\}, \Sigma, \delta_1 \cup \{ \delta(p, \lambda) = q_1, \delta(f_1, \lambda) = p \}, \{p\}, \{p\})$;

г) дополнения:

язык $P = \Sigma^* — L = \{ \alpha \mid \alpha \notin L \}$ является регулярным;

доказательство: язык P распознается конечным автоматом

$(Q_1, \Sigma, \delta_1, \{q_1\}, Q_1 — \{f_1\})$;

Автомат принимает дополнение языка, если все не допускающие состояния сделать допускающими и наоборот. При этом автомат должен быть полным, иначе часть цепочек не достигнет никакого состояния.

Чтобы определить дополняющий язык, нужно смоделировать собственно язык из выражения в автомат, поменять допускающие состояния и получить новое выражение из автомата.

д) пересечения:

язык $P = L \cap M = \{\alpha \mid \alpha \in L, \alpha \in M \}$ является регулярным;

пересечение выражается через объединение и дополнение:

$L \cap M = \Sigma^* — ((\Sigma^* — L) \cup (\Sigma^* — M))$ (по закону де-Моргана);

Пусть $\Sigma = \{a, b\}$, $L = a^*b^*$, $M = b^*$, тогда $L \cap M = b^*$.

е) разности:

язык $P = L — M = \{ \alpha \mid \alpha \in L, \alpha \notin M \}$ является регулярным;

разность выражается через пересечение и дополнение:

$L — M = L \cap (\Sigma^* — M)$;

Пусть $\Sigma = \{a, b\}$, $L = a^*b^*$, $M = b^*$, тогда $L — M = a^*$.

ж) обращения:

язык $P = L^R = \{ \alpha^R \mid \alpha \in L \}$ является регулярным;

доказательство простое: язык P распознается обращенным конечным автоматом.

Пусть $\Sigma = \{a, b\}$, $L = a^*b^*$, тогда $L^R = b^*a^*$.

2.10.3. Гомоморфизмы

Операция гомоморфизма формализует идею посимвольного перевода слов одного алфавита в слова другого.

Пусть Σ и Θ — два алфавита. Отображение h: $\Sigma^* \to \Theta^*$ слов первого алфавита в слова второго называется *гомоморфизмом*, если

1) $h(\lambda) = \lambda$;

2) для любых двух слов w_1 и w_2 в алфавите Σ имеет место равенство $h(w_1w_2) = h(w_1)h(w_2)$.

Из этого определения непосредственно следует, что гомоморфизм однозначно определяется своими значениями на символах алфавита Σ:

если $w = w_1w_2\ldots w_n$, $w_i \in \Sigma$, $1 \le i \le n$, то $h(w) = h(w_1)h(w_2)\ldots h(w_n)$.

Гомоморфизм можно представить как функцию, определенную на множестве цепочек, которая подставляет определенную цепочку *вместо каждого символа* данной цепочки.

Пример гомоморфизма

Пусть $\Sigma = \{a, b, c\}$, $\Theta = \{\mathbf{0}, \mathbf{1}\}$, и гомоморфизм h определен на символах алфавита Σ следующим образом: $h(a) = \mathbf{00}$, $h(b) = \mathbf{11}$, $h(c) = \lambda$.

Тогда $h(aba) = \mathbf{001100}$, $h(acbc) = \mathbf{0011}$, $h(cc) = \lambda$.

Если $L = a^*b^*c$, тогда $h(L) = (\mathbf{00})^*(\mathbf{11})^*$. ●

Пусть h: $\Sigma^* \to \Theta^*$ — произвольный гомоморфизм, а L — язык в алфавите Σ: $L(\Sigma)$. Гомоморфизм языка L — это язык $h(L) = \{ h(w) \mid w \in L \}$, определяемый как множество цепочек языка L, к которым был применен гомоморфизм h. Цепочку $h(w)$ называют также *образом* цепочки w, а язык $h(L)$ называют *образом* языка L при гомоморфизме h. Тогда образ языка L есть множество образов всех цепочек L при гомоморфизме h.

Определим понятие обратного гомоморфизма.

Пусть L — язык в алфавите Σ. *Прообразом* языка L при гомоморфизме h называется язык $h^{-1}(L) = \{ w \in \Sigma^* \mid h(w) \in L \}$, состоящий из цепочек в алфавите Σ, образы которых при гомоморфизме h принадлежат L.

Пример обратного гомоморфизма

Пусть $\Sigma = \{a, b\}$, $\Theta = \{0, 1\}$, и гомоморфизм h определен на символах алфавита Σ следующим образом: $h(a) = \mathbf{00}$, $h(b) = \mathbf{1}$.

Пусть $L = \mathbf{0011}^*$, то есть два нуля в начале цепочек и любое число единиц далее. Тогда язык $h^{-1}(L) = abb^*$, так как $abb^* \in \Sigma^*$ и $h(abb^*) \in L$. ●

Доказано, что класс регулярных языков замкнут как относительно гомоморфизма, так и относительно обращения гомоморфизма (взятия прообраза).

Эти формальные доказательства сложные, см., например, [11].

2.10.4. Свойства разрешимости

Для регулярных языков разрешимы проблемы, неразрешимые для других типов языков. Например, доказано, что разрешимыми являются:

- проблема *принадлежности* цепочки языку;
- проблема *пустоты* языка;
- проблема доказательства *эквивалентности* двух языков.

Проблема принадлежности цепочки языку может быть решена следующим образом: если некоторый конечный автомат M для произвольной цепочки w переходит из начального состояния в одно из финальных, то цепочка w принадлежит языку $L(M)$.

Проблема пустоты языка решается так: если для некоторого конечного автомата M множество достижимых состояний содержит любое из допускающих состояний, то автомат порождает непустую цепочку.

Доказательство эквивалентности двух языков сводится к доказательству эквивалентности двух ДКА, эквивалентных этим языкам. Для этого необходимо построить эквивалентные языкам λ-НКА и минимизировать их методом Бржозовского.

В качестве примера рассмотрим два языка, регулярные выражения для которых равны $(ab)^*a$ и $a(ba)^*$ (пункт 17 свойств регулярных выражений). Соответствующие им λ-НКА приведены на следующем рисунке.

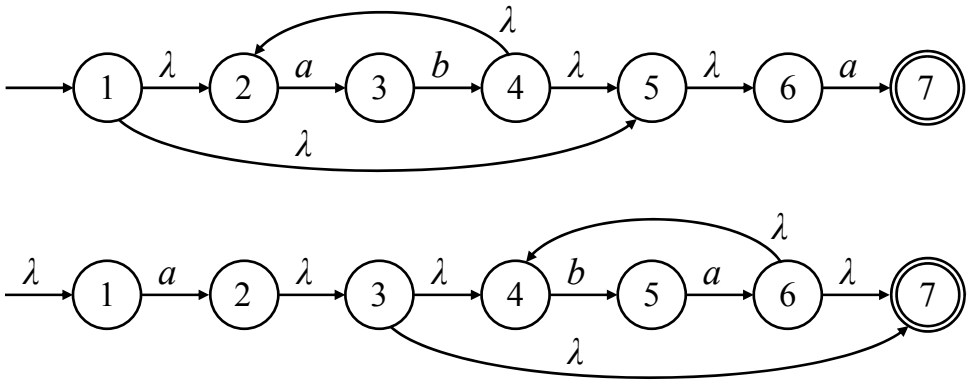


Рисунок 2.22. НКА, эквивалентные выражениям $(ab)^*a$ и $a(ba)^*$

Минимизация этих λ-НКА приводит к одному и тому же ДКА, граф переходов которого показан на следующем рисунке.

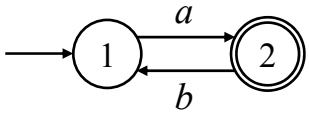


Рисунок 2.23. Минимальные ДКА для выражений $(ab)^*a$ и $a(ba)^*$

Следовательно, данные выражения являются эквивалентными.

2.11. Вопросы и упражнения

Вопросы для самопроверки

1. Опишите цели лексического анализа и его приложения.
2. Опишите лексемы языков программирования.
3. Дайте определение регулярным операциям.
4. Дайте определение регулярного множества.
5. Дайте определение регулярного выражения.
6. Как соотносятся регулярные множества и выражения?
7. Дайте определение регулярного языка.
8. Каков приоритет регулярных операций?
9. Дайте определение конечного автомата.
10. Опишите назначение функции переходов.
11. Что означают записи переходов $p \in \delta(q, a)$ и $\delta(q, a) = \{p_1, p_2\}$?
12. Что называется конфигурацией конечного автомата?
13. Какая конфигурация является начальной, а какая допускающей?
14. Что называется тактом работы конечного автомата?
15. Что такое путь конечного автомата? Какой путь успешен?
16. Приведите формулу языка, допускаемого конечным автоматом.
17. Дайте определение графу переходов конечного автомата.
18. Дайте определение таблице переходов конечного автомата.
19. Что называется полностью определенным конечным автоматом?
20. Как привести конечный автомат к полному виду?
21. Дайте определение ДКА и НКА.
22. Что такое конструкция подмножеств и как она выполняется?
23. Опишите алгоритм приведения НКА к ДКА.
24. Докажите эквивалентность НКА к ДКА.
25. Опишите последовательность построения конечного автомата по регулярному выражению.
26. Что такое λ-переход, *λ-НКА*, функция *closure*(q)?
27. Опишите метод получения регулярного выражения по конечному автомату.
28. Опишите конструкцию право- (лево-) линейной грамматики.
29. Докажите эквивалентность право- (лево-) линейных автоматных грамматик конечным автоматам.
30. Какой вид имеют уравнения с регулярными коэффициентами? Что является решением таких уравнений?
31. Как построить систему уравнений с регулярными выражениями? Как решается такая система?
32. Опишите метод преобразования регулярного выражения в регулярную грамматику.

33. Какие состояния конечного автомата являются эквивалентными, неэквивалентными?

34. Как построить таблицу неэквивалентности состояний?

35. В чем заключается минимизация конечного автомата?

36. Дайте определение левым и правым языкам состояний автомата.

37. Дайте определения детерминированного и минимального конечного автомата, используя понятия левых и правых языков состояний.

38. Дайте определение обращенного конечного автомата.

39. Опишите минимизацию по методу Бржозовского.

40. Опишите лемму о разрастании для регулярных языков.

41. Опишите свойства замкнутости регулярных языков.

42. Что такое гомоморфизм, обратный гомоморфизм?

43. Как доказывается проблема пустоты, принадлежности?

44. Как доказывается эквивалентность регулярных языков?

Упражнения

1. Опишите языки, определяемые регулярными выражениями:

а) $(a+b)a^*$;

б) $(a+b+1)a^*$;

в) $(a+b)(a+b+1)^*$;

г) $ab(a^*+b^*)cc^*$;

д) $a^*bb^*c^*$;

е) $aa^*babb^*cc^*$;

ж) $(ab+c)^*c^*$;

2. Задайте регулярные выражения для следующих языков:

а) цепочки вида: буква, цифра, цифра …;

б) цепочки вида: буква, две цифры, буква, две цифры, …;

в) цепочек вида две буквы, четыре буквы, шесть букв, …;

г) целых чисел со знаком "+" или "–" или без знака;

д) вещественной константы в форматах "0.", ".0" и "0.0".

3. Преобразуйте в регулярную грамматику и конечный автомат регулярные выражения из п.1 и п.2.

4. Преобразуйте в регулярное выражение и конечный автомат регулярную грамматику:

а) $S\to aA \mid bB,\ A\to b \mid aB,\ B\to a \mid bA$;

б) $H\to aA \mid aB,\ A\to a \mid b \mid aA \mid aB,\ B\to a \mid b \mid aA \mid aB$;

5. Преобразуйте грамматику в автомат и детерминируйте его:

а) $S\to aS \mid bS \mid aA,\ A\to aB,\ B\to a \mid b$;

б) грамматика из п.4*б*.

6. Минимизируйте автомат методами Хопкрофта и Бржозовского:

а) автомат М3 (рисунок 2.14);

б) автомат, полученный в п.5*б*.

2.12. Литература

Лучшим источником информации по лексическому анализу является основательный первоисточник Хопкрофт [11]. Многократно всеми списываемый первоисточник Ахо [3] также нельзя обойти вниманием. Для написания этой главы использованы эти две книги, [4][7][9][12], другие источники тщательно изучались и понемногу вносили свою лепту.

Разделы о лексемах и их распознавании являются оригинальными, переход конечный автомат ⇔ грамматика изложен в авторской версии.

Концепция левых и правых языков состояний конечного автомата заимствована из исследования [2]. Там же можно найти алгоритм минимизации Бржозовского.

1. Donald E. Knuth. Semantics of Context-Free Languages. Mathematical Systems Theory, 1968, Vol 2, No. 9, p.p.127-145.

2. J. M. Champarnaud, A. Khorsi, T. Paranthoën. Split and join for minimizing: Brzozowski's algorithm. The Prague Stringology Conference, 2002.

3. А. Ахо, Дж. Ульман. Теория синтаксического анализа, перевода и компиляции. Том 1. Синтаксический анализ. Пер. с англ. М.: Мир, 1978.

4. Ахо, Альфред, В., Сети, Рави, Ульман, Джеффри, Д. Компиляторы: принципы, технологии и инструменты.: Пер. с англ. — М.: Издательский дом «Вильямс», 2003. — 768 с.: ил.

5. Карпов Ю. Г. Теория и технология программирования. Основы построения трансляторов. — СПб.: БХВ-Петербург, 2005. — 272 с.: ил.

6. Компаниец Р.И., Маньков Е.В., Филатов Н.Е. Системное программирование. Основы построения трансляторов. / Учебное пособие для высших и средних учебных заведений. — СПб.: КОРОНА принт, 2000. — 256 с.

7. Опалева Э.А., Самойленко В.П. Языки программирования и методы трансляции. — СПб.: БХВ-Петербург, 2005. — 480 с.: ил.

8. Пентус А.Е., Пентус М.Р. Теория формальных языков: Учебное пособие. — М.: Изд-во ЦПИ при механико-математическом ф-те МГУ, 2004. — 80 с.

9. Рейуорд-Смит В. Дж. Теория формальных языков. Вводный курс: Пер. с англ. — М.: Радио и связь, 1988. — 128 с.: ил.

10. Системное программное обеспечение: Учебник для вузов / А. Ю. Молчанов. — СПб.: Питер, 2003. — 396 с.: ил.

11. Хопкрофт, Джон, Э., Мотвани, Раджив, Ульман, Джеффри, Д. Введение в теорию автоматов, языков и вычислений, 2-е изд. : Пер. с англ. — М.: Издательский дом «Вильямс», 2002. — 528 с.: ил.

12. Ф. Льюис, Д. Розенкранц, Р. Стирнз. Теоретические основы проектирования компиляторов: Пер. с англ. — М.: Мир, 1979. — 656 с.: ил.